2024-2
초급 스터디

2. 정수론

목차

모듈러 연산

- 어떤 수를 다른 수로 나눈 나머지를 구하는 연산
- $a \,\%\, b = c \quad \rightarrow \quad a = b \times Q + c$

모듈러 연산

모듈러 연산은 다음과 같은 성질을 갖는다.

$$(a + b) \,\%\, m = \big((a \,\%\, m) + (b \,\%\, m)\big) \,\%\, m$$
$$(a - b) \,\%\, m = \big((a \,\%\, m) - (b \,\%\, m)\big) \,\%\, m$$
$$(a \times b) \,\%\, m = \big((a \,\%\, m) \times (b \,\%\, m)\big) \,\%\, m$$

- 다 계산하고 나머지를 구한 결과와
  나머지를 취하고 계산한 뒤 나머지를 구한 결과가 같다.

모듈러 연산

- $a \,\%\, m = a \; - m \times k_1$
- $b \,\%\, m = b \; - m \times k_2$

- $(\boldsymbol{a} \,\%\, \boldsymbol{m}) + (\boldsymbol{b} \,\%\, \boldsymbol{m})$

$$= a + b \; - m\,(k_1 + \; k_2)$$
$$= a + b \; - m \times k_3$$
$$= (\boldsymbol{a} + \boldsymbol{b}) \,\%\, \boldsymbol{m}$$

모듈러 연산

- 위에서 정리한 등식 양변을 m으로 나눈 나머지를 구하면
- $\big((a \,\%\, m) + (b \,\%\, m)\big) \,\%\, m$

$$= \big((a + b) \,\%\, m\big) \,\%\, m$$

$$= (a + b) \,\%\, m$$

모듈러 연산

- 이 특징을 이용하여 나머지를 출력하는 문제를 풀 수 있다.

입력

첫 번째 줄에 자연수 N이 주어진다. ($1 \le N \le 1,000,000$)

출력

첫 번째 줄에 지원이가 만들 수 있는 길이가 N인 모든 2진 수열의 개수를 15746으로 나눈 나머지를 출력한다.

모듈러 연산

- 나머지를 출력하는 문제의 일반적인 특징은,
  나머지를 취하지 않은 답이 매우 크다는 것이다.
- 이 말은 이 답을 구하는 과정에서도 매우 큰 수의 연산을 반복해야 함을 의미한다.

모듈러 연산

- C와 같은 언어는 자료형 크기 제한으로 인해 큰 수를 쓸 수 없다.
- 파이썬은 수의 표현 범위 제한이 없지만, 큰 수 연산은 시간이 많이 걸린다.

```python
import datetime

print(datetime.datetime.now())
for _ in range(10**7):
    a = 11**18 * 11**18
print(datetime.datetime.now())
```

```
C:\Users\Everdu\anaconda3\python.exe D:\PS\ps2.py
2024-09-29 14:43:44.667023
2024-09-29 14:43:45.018024

Process finished with exit code 0
```

모듈러 연산

- 이때 만약 정답을 a로 나눈 나머지를 구해야 한다면,
  두 수를 연산하는 대신 **각각을 a로 나눈 나머지를 연산**하면
  큰 수 연산을 하지 않고도 같은 정답을 구할 수 있다.

- 이 테크닉은 여러 문제에서 자주 사용되는 테크닉이니
  나머지를 답으로 구하는 문제를 보면 꼭 사용하자.

모듈러 연산

- 예시

```python
from datetime import datetime

print(datetime.now())
a = 1
MOD = 12345
for _ in range(10**5):
    tmp = 11**8
    a *= (tmp * tmp)

print(a % MOD)
print(datetime.now())
```

```
2024-09-30 10:59:32.053136
4786
2024-09-30 11:00:02.447728
```

모듈러 연산

- 예시

```python
from datetime import datetime

print(datetime.now())
a = 1
MOD = 12345
for _ in range(10**5):
    tmp = 11**8
    tmp %= MOD
    a *= (tmp * tmp)
    a %= MOD

print(a)
print(datetime.now())
```

```
2024-09-30 11:01:22.556647
4786
2024-09-30 11:01:22.602202
```

GCD

- **약수** (Divisor) - 어떤 수를 나누어 떨어지게 하는 수
- 12의 약수 : 1, 2, 3, 4, 6, 12

약수를 구하는 방법

1부터 N까지 모든 자연수로 N을 나눠보고
이 중에 나누어 떨어지는 수들을 찾는다.
→ O(n)

```
n = 12

for i in range(1, n+1):
    if n % i == 0:
        print(i)
```

```
1
2
3
4
6
12
```

GCD

- 최대 공약수 (Greatest Common Divisor)
- 두 수가 공통으로 갖는 약수(공약수) 중 가장 큰 약수

- 12의 약수 : 1, 2, 3, 4, 6, 12
- 18의 약수 : 1, 2, 3, 6, 9, 18
- 최대 공약수 : GCD(12, 18) = 6

GCD

GCD는 다음과 같은 성질을 갖는다.

- $gcd(a, b, c) = gcd\big(a,\ gcd(b, c)\big)$
- $gcd(a, b, c, \dots) = gcd(a,\ gcd(b, gcd(c, \dots)))$

GCD

gcd(12, 18, 27)

= gcd(gcd(12, 18), 27)

= gcd(6, 27)

= 3

GCD를 구하는 방법

첫번째 방법

- 두 수의 모든 약수를 구한 뒤, 가장 큰 공약수를 찾는다.

- 시간복잡도: $O(max(a, b))$
  두 수 중에 1억이 하나만 있어도 1초가 소요된다.

GCD를 구하는 방법

두번째 방법

- **유클리드 호제법**으로 GCD를 계산한다.

유클리드 호제법

- $\mathbf{r = a \% b}$ 일 때, $\mathbf{gcd}(a,\ b) = \mathbf{gcd}(b,\ r)$ 가 성립한다.
- 이 성질을 반복적으로 적용하면
  r = 0 이 될 때의 b가 최대 공약수가 된다.

- 유클리드 호제법의 시간 복잡도 : $O(log(max(a, b)))$

유클리드 호제법

- $\gcd(1071, 1029)$
  $= \gcd(1029, 42)$
  $= \gcd(42, 21)$
  $= \gcd(21, 0)$
- 최대 공약수는 21

유클리드 호제법

- 구현

```python
def gcd(a, b):
    print(a, b)
    if b == 0:
        return a

    return gcd(b, a%b)

print(gcd( a: 1071,  b: 1029))
```

```
1071 1029
1029 42
42 21
21 0
21
```

유클리드 호제법

- math 모듈에 있는 gcd 함수를 사용할 수도 있다.

```
from math import gcd

print(gcd(1071, 1029))
```

```
C:\Users\kckc0\AppData\Local\Progr
21

Process finished with exit code 0
```

LCM

- 최소 공배수 (Least Common Multiple)
- 두 수의 공통된 배수 중 가장 작은 수

LCM

- LCM은 두 수를 GCD로 나눈 각각의 몫과 GCD의 곱과 같다. 따라서 다음이 성립한다.
- $\boldsymbol{a \times b = \mathbf{gcd}(a,b) \times lcm(a,b)}$

LCM

증명

- $a = gcd(a,b) \times a',\ b = gcd(a,b) \times b'$ (이때 a', b' 은 서로소)
- $a \times b = gcd(a,b) \times gcd(a,b) \times a' \times b'$

$= gcd(a,b) \times LCM(a,b)$

LCM

- 따라서 LCM은 $a \times b \div gcd(a, b)$ 으로 구할 수 있다.

  (또는 Math 모듈에 있는 lcm 함수를 쓸 수도 있다.)

연습 문제

- https://www.acmicpc.net/problem/13241
- 입력으로 들어오는 수가 최대 1억이라
  공약수를 구할 때 **유클리드 호제법을 사용**해야 한다.

연습 문제

- 직접 풀어봅시다.

연습 문제

- 정답 코드

```python
from math import gcd

a, b = map(int, input().split())
g = gcd(a, b)
print(a*b//g)
```

```python
from math import lcm

a, b = map(int, input().split())
print(lcm(a, b))
```

연습 문제

- https://www.acmicpc.net/problem/2485
- 같은 간격으로 가로수를 심을 때, 최소 가로수의 수 구하기

연습 문제

- 나무를 최소한으로 심으려면 나무와 나무 사이 **간격을 최대**로 늘려야 한다.

연습 문제

- 첫 번째 나무의 위치를 a, 간격을 d 라고 하면
  i 번째 나무의 위치는 $\boldsymbol{a + d \times (i - 1)}$ 로 표현할 수 있다.
- 이때, 나무의 위치에서 a를 빼면
  모든 나무의 위치는 **d의 배수**이다. (즉, d를 **약수**로 갖는다)
- d 값을 최대로 키우려면 GCD를 구하면 된다!

연습 문제

- 직접 풀어봅시다.

연습 문제

- 정답 코드

```
import sys
from math import gcd
input = sys.stdin.readline

n = int(input())
first = int(input())
trees = []
```

```
for _ in range(n-1):
    tree = int(input())
    trees.append(tree - first)

g = gcd(trees[0], trees[1])
for i in range(2, len(trees)):
    g = gcd(g, trees[i])

print(trees[-1]//g - n + 1)
```

연습 문제

- 정답 코드 (리스트 축약)

```python
import sys
from math import gcd
input = sys.stdin.readline

n = int(input())
first = int(input())
pos = [int(input()) - first for _ in range(n-1)]
g = gcd(pos[1], pos[0])
for i in range(2, len(pos)):
    g = gcd(g, pos[i])

print(pos[-1]//g - n + 1)
```

소수

- 소수 : 1과 자기 자신만을 약수로 갖는 수
- 합성수 : 1과 자기 자신 외에 약수를 갖는 수

소수 판정

- 모든 약수를 다 구해서 1과 자기 자신 외의 약수가 있는지 확인
  → O(n)

소수 판정

- 1부터 N 까지 수 중에 모든 소수 판정하기
  → 이전과 같은 방식을 사용하면 $O(N^2)$

- 더 빠른 방법은 없을까?

소수 판정 - 에라토스테네스의 체

- 특정 범위에 있는 모든 소수를 찾는 알고리즘

소수 판정 - 에라토스테네스의 체

- 2부터 찾는 범위 N까지 수를 모두 나열한다.
- 2부터 N까지 다음 단계를 반복하여 수행한다.
  - 만약 현재 수 p가 지워지지 않은 수라면 p는 소수이다.
    p ~ N 사이 모든 p의 배수를 지운다.
  - 만약 현재 수 p가 지워진 수라면 합성수이므로 넘어간다.

소수 판정 - 에라토스테네스의 체

	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

소수 판정 - 에라토스테네스의 체

	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

소수 판정 - 에라토스테네스의 체

	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

소수 판정 - 에라토스테네스의 체

	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

소수 판정 - 에라토스테네스의 체

	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

에라토스테네스의 체

- 구현

```python
n = 72
is_prime = [True]*(n+1)
is_prime[0] = False
is_prime[1] = False
for i in range(2, n+1):
    if is_prime[i]:
        for j in range(i+i, n+1, i):
            is_prime[j] = False

for i in range(2, n+1):
    if is_prime[i]:
        print(i, end=' ')
```

```
C:\Users\kckc0\AppData\Local\Programs\Python\Python38\pyth
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71
Process finished with exit code 0
```

에라토스테네스의 체

- N이 소수인지 판정할 때 2부터 N까지 모두 확인할 필요는 없다.
- 어떤 수의 모든 약수는 $\sqrt{N}$ 을 기준으로 대칭이다.

- 36의 약수 : 1 2 3 4 6 9 12 18 36

에라토스테네스의 체

- 따라서 합성수 N은 **2이상 $\sqrt{N}$ 이하**의 약수를 반드시 갖고 있다.
- $2 \leq x \leq \sqrt{N}$ 인 모든 x 에 대해 배수를 제거하고 나면 $\sqrt{N} < k \leq N$ 인 모든 k 에 대해 합성수인 k 는 제거 된다.
  합성수인 k 는 반드시 **$\sqrt{k}$ 이하의 약수**를 갖고 있기 때문이다.

에라토스테네스의 체

- 개선된 구현

```
n = 72
is_prime = [True]*(n+1)
is_prime[0] = False
is_prime[1] = False
for i in range(2, int(n**0.5)+1):
    if is_prime[i]:
        for j in range(i+i, n+1, i):
            is_prime[j] = False

for i in range(2, n+1):
    if is_prime[i]:
        print(i, end=' ')
```

에라토스테네스의 체

+) 어떤 수가 소수인지 판정할 때도 $O(\sqrt{N})$ 에 구할 수 있다.

모든 약수를 구해볼 필요가 없기 때문이다.

```
x = int(input())
for i in range(2, int(x**0.5)+1):
    if x % i == 0:
        print("not prime")
        exit(0)

print("prime")
```

에라토스테네스의 체

- 배수를 지울 때도 i + i 가 아니라 i * i 부터 지울 수 있다.
- i 미만의 수에 대해 배수를 지우면서 i *2 부터 i * (i-1) 까지는 이미 지웠기 때문이다.

에라토스테네스의 체

- 개선된 구현

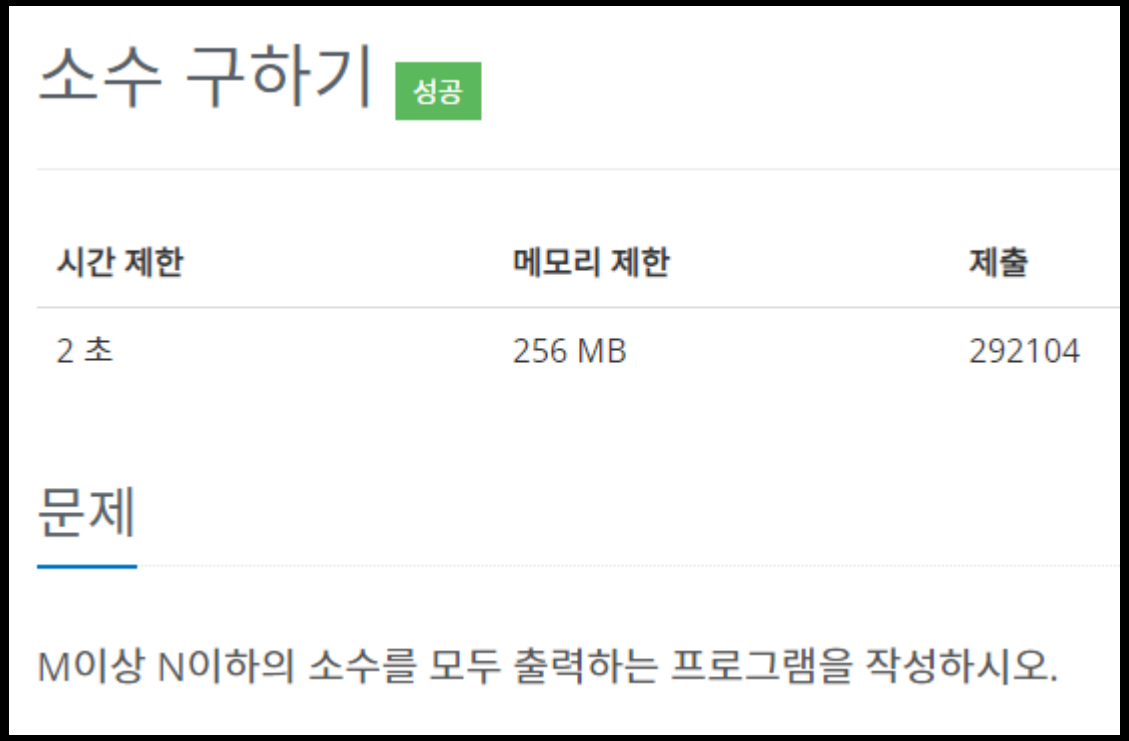
```
n = 72
is_prime = [True]*(n+1)
is_prime[0] = False
is_prime[1] = False
for i in range(2, int(n**0.5)+1):
    if is_prime[i]:
        for j in range(i*i, n+1, i):
            is_prime[j] = False

for i in range(2, n+1):
    if is_prime[i]:
        print(i, end=' ')
```

에라토스테네스의 체

- 시간 복잡도 : O(n log log n)

연습 문제

- https://www.acmicpc.net/problem/1929



소수 구하기 성공

시간 제한	메모리 제한	제출
2 초	256 MB	292104

문제

M이상 N이하의 소수를 모두 출력하는 프로그램을 작성하시오.

연습 문제

- N ≤ 100만 이므로 1~100만까지 소수를 모두 구해둔 뒤, M부터 돌면서 소수 여부를 확인하자.

연습 문제

- 직접 풀어봅시다.

연습 문제

- 정답 코드

```python
m, n = map(int, input().split())
is_prime = [True]*(n+1)
is_prime[0] = False
is_prime[1] = False
for i in range(2, int(n**0.5)+1):
    if is_prime[i]:
        for j in range(i*i, n+1, i):
            is_prime[j] = False

for i in range(m, n+1):
    if is_prime[i]:
        print(i)
```

연습 문제

- https://www.acmicpc.net/problem/6588

골드바흐의 추측 성공 다국어

시간 제한	메모리 제한	제출
0.5 초	256 MB	91336

문제

1742년, 독일의 아마추어 수학가 크리스티안 골드바흐는 레온하르트 오일러에게 다

> 4보다 큰 모든 짝수는 두 홀수 소수의 합으로 나타낼 수 있다.

예를 들어 8은 3 + 5로 나타낼 수 있고, 3과 5는 모두 홀수인 소수이다. 또, 20 = 3 +

이 추측은 아직도 해결되지 않은 문제이다.

백만 이하의 모든 짝수에 대해서, 이 추측을 검증하는 프로그램을 작성하시오.

연습 문제

- 각 테스트케이스마다 매번 소수를 구하는 것보다,
  미리 소수를 다 구해두고 구해둔 소수를 활용하는 것이 더 효율적이다.

연습 문제

- 한번 풀어봅시다.

연습 문제

- 정답 코드

```python
import sys
input = sys.stdin.readline

n = 2000000
is_prime = [True]*(n+1)
prime_list = []
for i in range(2, int(n**0.5)+1):
    if is_prime[i]:
        for j in range(i*i, n+1, i):
            is_prime[j] = False

for i in range(2, n+1):
    if is_prime[i]:
        prime_list.append(i)
```

```python
while True:
    n = int(input())
    if n == 0:
        break

    find = False
    for num in prime_list:
        if n > num and is_prime[n - num]:
            print(f"{n} = {num} + {n-num}")
            find = True
            break

    if not find:
        print("Goldbach's conjecture is wrong.")
```

정리

- **모듈러 연산** : 특정 수로 나눈 나머지를 구할 때, 분배법칙 이용
- **최대 공약수** : 유클리드 호제법, 최소 공배수
- **에라토스테네스의 체** : 특정 범위의 모든 소수 찾기

수고하셨습니다 ☺

스터디 관련 의견을 남겨주시면 다음 주에 반영하겠습니다!

- https://forms.gle/hMfmWvTbBnhJp24G6