
# 실전 알고리즘 0x0B강

# 수학

BaaaaaaaaaaaaaaaarkingDog


# 목차

# 시작하기에 앞서..


- 이번 강의에서 다루는 수학은 코딩테스트를 통과하기 위한 정도의 수학일 뿐이지 앞으로 프로그래밍을 할 때 이 정도만 알고 있으면 충분하다는 뜻이 절대 아닙니다.
- 컴퓨터 공학 학부 과정에서는 보통 이산수학, 선형대수학, 미적분학, 확률과 통계 정도는 배웁니다. 물론 어떤 분야에 일하느냐에 따라 수학이 쓰이는 정도가 다르겠지만 위의 네 과목 정도는 기본기로 가지고 있는 것이 좋습니다.
- 엄밀한 것을 좋아하는 분들을 위해 이번 강의에서는 각 알고리즘이 왜 성립하는지를 꽤 상세하게 다루고 있습니다. 수학적 증명에 익숙하지 않으면 잘 이해가 가지 않을 수도 있는데, 증명이 크게 중요한 것은 아니니 잘 이해가 가지 않으면 그냥 받아들이고 사용하셔도 아무 상관 없습니다.

# 0x00 소수 – 소수 판정법

- 소수는 1과 자신으로만 나누어 떨어지는 수입니다.
- $N$이 소수인지 판단하는 함수를 만들어봅시다. 가장 쉬운 방법은 2부터 $N$-1중에 $N$을 나누는 수가 있는지 판단하는 방법입니다. 지금까지의 과정을 잘 따라왔다면 굉장히 쉽게 구현할 수 있을 것입니다. 이 방법의 시간복잡도는 O($N$)입니다.

```
bool isPrime1(int n){
  if(n==1) return false;
  for(int i = 2; i < n; i++){
    if(n%i == 0) return false;
  }
  return true;
}
```

# 0x00 소수 – 소수 판정법


- 그런데 수학적 관찰을 한 가지 하게 된다면 시간복잡도를 $O(\sqrt{N})$ 으로 떨굴 수 있습니다.
- 이전에 작성했던 코드를 생각해보면 결국 1과 $N$을 제외한 $N$의 약수 중에서 제일 작은 $i$ 를 만날 때 바로 탈출합니다. 그런데 $i$ 가 $\sqrt{N}$ 보다 클 수 있을까요? 그렇지 않음을 증명해봅시다.

# 0x00 소수 – 소수 판정법


- 명제 : 1과 $N$을 제외한 $N$의 약수가 존재할 때, 그 중에서 최솟값을 a라고 하자. 이 때 $a \leq \sqrt{N}$ 이다.

- 증명 : $a > \sqrt{N}$ 이라고 가정하고 모순을 찾아보자. 우선 $b = N/a$ 가 $N$의 약수임은 자명하다. 1과 $N$을 제외한 $N$의 약수 중에서 최솟값이 a이므로 b는 a보다 크거나 같아야한다. 그런데 $a > \sqrt{N}$ 이므로 $b = N/a < \sqrt{N}$ 이기에 모순이다. 그러므로 $a \leq \sqrt{N}$ 이다.

# 0x00 소수 – 소수 판정법

- 앞에서 증명한 명제로부터 i 를 2부터 $N$-1까지 돌릴 필요 없이 $i^2$ 가 $N$ 이하일 때 까지 돌리면 됨을 알 수 있습니다. 이를 통해 시간복잡도를 O($\sqrt{N}$) 으로 줄일 수 있습니다.

```
bool isPrime2(int n){
  if(n==1) return false;
  for(int i = 2; i*i <= n; i++){
    if(n%i == 0) return false;
  }
  return true;
}
```

cmath 헤더에 `sqrt` 함수가 존재하지만 해당 함수는 실수를 인자로 받는 함수이기 때문에 실수오차가 발생할 수 있습니다. 그렇기 때문에 `i <= sqrt(n)` 대신 `i*i <= n`을 써야 합니다.

# 0x00 소수 – 에라토스테네스의 체

- 앞의 소수 판정법은 $N$이 소수인지를 판단하는 알고리즘입니다. $N$이하의 소수를 모두 찾는 알고리즘은 어떻게 만들 수 있을까요?
- 가장 간단한 방법은 이전의 소수 판정법을 1부터 $N$까지의 모든 수에 대해 다 해보는 것입니다. $N$ = 20,000,000일 때 BOJ 서버 기준으로 0.15초 정도 걸리기에 굉장히 효율적인 방법 중 하나입니다. (그러나 함수 호출이 꽤 시간이 오래 걸리는 연산이기 때문에 우측의 코드는 $N$ = 5,000,000일 때 1.87초가 걸렸습니다.)

```
vector<int> allPrime1(int n){
  vector<int> ret;
  for(int i = 1; i <= n; i++){
    if(i==1) continue;
    for(int j = 2; j*j <= i; j++)
      if(i%j == 0) continue;
    ret.push_back(i);
  }
  return ret;
}
```

```
vector<int> allPrime1(int n){
  vector<int> ret;
  for(int i = 1; i <= n; i++){
    if(isPrime2(i)) ret.push_back(i);
  }
  return ret;
}
```


# 0x00 소수 – 에라토스테네스의 체

- 그러므로 $N$이하의 소수를 모두 찾기 위해 그냥 소수 판정법을 $N$번 돌려도 아무 상관 없지만 에라토스테네스의 체라는, $N$이하의 소수를 모두 찾는 간단한 알고리즘을 소개해드리겠습니다. ($N$ = 20,000,000일 때 BOJ 서버 기준으로 0.13초가 걸렸기 때문에 앞의 방법과 속도는 거의 차이가 없으나 메모리를 13~40배 정도 절약할 수 있습니다.)

# 0x00 소수 – 에라토스테네스의 체

1. $N$칸짜리 배열을 만듭니다. 이 배열은 해당 칸의 수가 소수일 경우 `true`, 소수가 아닐 경우 `false`를 의미합니다. 우선 1을 나타내는 칸은 `false`, 나머지는 `true`로 초기화시킵니다.

공간이 부족해 2차원 배열로 나타냈지만 원래는 1차원 배열입니다.

| | | | | | |
|---|---|---|---|---|---|
| 1 | 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 |

# 0x00 소수 – 에라토스테네스의 체

2. 커서를 하나 두어 2를 가리키게끔 합니다. 해당 커서는 2부터 $N$까지 진행하면서 소수를 찾은 후 뭔가 작업을 할 것입니다.

| | | | | | |
|---|---|---|---|---|---|
| 1 | 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 |

# 0x00 소수 – 에라토스테네스의 체

3. 현재 커서가 2를 가리키고 있고 해당 칸은 `true`이므로 2는 내버려두고 2가 아닌 모든 2의 배수들을 `false`로 만듭니다.

| | | | | | |
|---|---|---|---|---|---|
| 1 | 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 |

# 0x00 소수 – 에라토스테네스의 체


4. 커서를 다음 칸으로 옮깁니다. 현재 커서가 3를 가리키고 있고 해당 칸은 `true`이므로 3는 내버려두고 3가 아닌 모든 3의 배수들을 `false`로 만듭니다.

| | | | | | |
|---|---|---|---|---|---|
| 1 | 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 |

# 0x00 소수 – 에라토스테네스의 체


5. 커서를 다음 칸으로 옮깁니다. 현재 커서가 4를 가리키고 있고 해당 칸은 `false`이므로 아무 작업도 하지 않고 넘어갑니다.

| | | | | | |
|---|---|---|---|---|---|
| 1 | 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 |

# 0x00 소수 – 에라토스테네스의 체

6. 커서를 다음 칸으로 옮깁니다. 현재 커서가 5를 가리키고 있고 해당 칸은 `true`이므로 5는 내버려두고 5가 아닌 모든 5의 배수들을 `false`로 만듭니다.

| | | | | | |
|---|---|---|---|---|---|
| 1 | 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 |

# 0x00 소수 – 에라토스테네스의 체


7. 커서가 $N$에 도달할 때 까지 작업을 반복합니다.
작업이 끝난 뒤 `true`인 칸이 곧 소수를 의미합니다.

| | | | | | |
|---|---|---|---|---|---|
| 1 | 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 |

# 0x00 소수 – 에라토스테네스의 체


- 구현은 그다지 어렵지 않습니다.

```
vector<int> allPrime2(int n){
  vector<int> ret;
  int state[n+1];
  state[1] = 0;
  for(int i = 2; i <= n; i++) state[i] = 1;
  for(int i = 2; i <= n; i++){
    if(!state[i]) continue;
    for(int j = 2*i; j <= n; j += i) state[j] = 0;
  }
  for(int i = 1; i <= n; i++){
    if(state[i]) ret.push_back(i);
  }
  return ret;
}
```

# 0x00 소수 – 에라토스테네스의 체

- 이 구현은 최적화시킬 수 있는 여지가 세 군데 있습니다. 최적화를 시켜 속도와 메모리를 모두 향상시킵시다.

- **최적화 1** : 현재 코드에서는 `i = k`일 때 `2k,3k,4k,…`를 모두 `false`로 둔다. 그러나 `2k,3k,4k, … ,(k-1)k` 는 각각 2, 3, 4, ⋯ , `k-1`으로 나누어 떨어지기 때문에 이들은 `k`보다 더 작은 소인수가 존재한다. 그러므로 이들은 현재 `i = k` 에서 신경쓰지 않더라도 이전에 다른 `i` 값에서 이미 `false`로 바뀌어 있다. 그러므로 `2k,3k,4k, … ,(k-1)k` 에 대해서는 굳이 `false`로 바꿀 필요가 없는 것이고, `j`가 `2i`부터 시작할 필요 없이 $i^2$부터 시작하면 된다.

- **최적화 2** : 최적화 1에 따라 $i^2$이 $N$보다 커지면 더 이상 아무 값도 바꾸지 않으므로 $i^2$이 $N$이하일 때 까지만 반복문을 돌리면 된다.

# 0x00 소수 – 에라토스테네스의 체

- 최적화 3 : state 배열은 어차피 true 혹은 false만 저장하므로 원소 하나당 4byte 공간이나 잡아먹는 int로 두지 말고 GCC 기준 원소 하나당 1 bit씩만 잡는 vector<bool>로 잡으면 된다. (단 bool 배열은 원소 하나당 1byte 공간을 차지합니다.)

```
int arr1[160]; // 640byte
bool arr1[160]; // 160byte
vector<bool> arr2(160); // 160bit = 20byte
```

# 0x00 소수 – 에라토스테네스의 체

- 최적화를 끝내고 나면 $N$ = 20,000,000일 때 BOJ 서버 기준으로 0.63초에서 0.13초로 줄일 수 있습니다.

```
vector<int> allPrime3(int n){
  vector<int> ret;
  vector<bool> state(n+1,true);
  state[1] = false;
  for(int i = 2; i*i <= n; i++){
    if(!state[i]) continue;
    for(int j = i*i; j <= n; j += i) state[j] = false;
  }
  for(int i = 1; i <= n; i++){
    if(state[i]) ret.push_back(i);
  }
  return ret;
}
```

# 0x00 소수 – 에라토스테네스의 체

## <정리>

- $N$이하의 소수를 모두 찾고 싶을 때 수 각각에 대해 소수를 판별하는 방법을 사용해도 괜찮다.
- 에라토스테네스의 체를 사용해도 괜찮은데, 앞에서 언급한 3가지 최적화를 해주는 것이 좋다. 최적화를 하지 않는다면 차라리 에라토스테네스의 체를 쓰는 것 보다 수 각각에 대해 소수를 판별하는게 더 빠르게 동작한다.
- 최적화를 한 에라토스테네스의 체 혹은 수 각각에 대한 소수를 판별하는 방법은 50,000,000 이하의 소수를 대략 0.35초안에 찾고, 100,000,000 이하의 소수를 대략 0.72초 안에 찾는다. (정확한 시간복잡도는 O($N\lg\lg N$))

# 0x00 소수 – 소인수분해


- 산술의 기본 정리 – 2 이상의 모든 자연수는 소수의 곱으로 나타낼 수 있다. 순서를 고려하지 않을 경우 나타내는 방법은 유일하다. (60 = 2· 2· 3· 5)

- 이제 소인수분해를 직접 해봅시다.(BOJ 11653번 : 소인수분해)
- $N$을 소인수분해하기 위해 $N$이하의 모든 소수를 구한 후 그 소수들로 $N$을 나눠보면 되지만 그닥 추천하고 싶은 방법은 아닙니다.
- 대신 괜찮은 알고리즘을 소개해드리겠습니다.

# 0x00 소수 – 소인수분해

1. i를 2로 초기화시키고 소인수 목록을 담을 배열을 준비해둡니다.

| | |
|---|---|
| $N$ | 1100 |
| i | 2 |

소인수 목록

# 0x00 소수 – 소인수분해


2. $N$이 i로 나누어 떨어지는지 확인합니다.
1100은 2로 나누어 떨어지므로 소인수 목록에
2를 추가하고 $N$을 2로 나눕니다.

$N$ 550

i 2

소인수 목록

2

# 0x00 소수 – 소인수분해


3. $N$이 i로 나누어 떨어지지 않을 때 까지 이 작업을 반복합니다.

$N$ 275

i 2

소인수 목록

2 2

# 0x00 소수 – 소인수분해


4. 더 이상 $N$이 i로 나누어 떨어지지 않는다면 i를 1 증가시킵니다. 단 $N$이 1일 경우에는 알고리즘을 종료합니다.

$N$ 275

i 3

소인수 목록

2 2

# 0x00 소수 – 소인수분해

5. 275는 3으로 나누어 떨어지지 않으므로 i를 1 증가시킵니다.

$N$ 275

i 4

소인수 목록

2 2

# 0x00 소수 – 소인수분해

6. 275는 4로 나누어 떨어지지 않으므로 i를 1 증가시킵니다.

| | |
|---|---|
| $N$ | 275 |
| i | 5 |

소인수 목록

2 2

# 0x00 소수 – 소인수분해


7. 275는 5로 나누어 떨어집니다. 이전에 한 것과 마찬가지로 $N$이 i로 나누어 떨어지는 동안 소인수 목록에 i를 추가하고 $N$을 i로 나누는 연산을 반복합니다.

$N$ 11

i 5

소인수 목록

2 2 5 5

# 0x00 소수 – 소인수분해

8. 더 이상 $N$이 i로 나누어 떨어지지 않으므로 i를 1 증가시킵니다.

$N$ 11

i 6

소인수 목록

2 2 5 5

# 0x00 소수 – 소인수분해


9. 더 이상 $N$이 i로 나누어 떨어지지 않으므로 i를 1 증가시킵니다. (i가 7,8,9,10일 땐 건너뛰겠습니다.)

$N$: 11

i: 11

소인수 목록

2 2 5 5

# 0x00 소수 – 소인수분해

10. $N$이 i로 나누어 떨어지므로 소인수 목록에 i를 추가하고 $N$을 i로 나누는 연산을 반복합니다.

$N$ 1

i 11

소인수 목록

2 2 5 5 11

# 0x00 소수 – 소인수분해

11. $N$이 1이 되었으므로 알고리즘을 종료합니다.
$N$(=1100)의 모든 소인수를 찾는데
성공했습니다.

$N$ | 1

i | 11

소인수 목록

2 2 5 5 11

# 0x00 소수 – 소인수분해


- 이 알고리즘이 올바른 소인수분해 결과를 준다는 것을 알 수 있을까요? 생각해보면 $i$로 $N$이 나눠지는지 확인할 때 $i$가 소수인지를 체크하지 않는데 이 알고리즘은 올바르게 동작하는걸까요?
- 우선 알고리즘이 반드시 종료됨을 증명해야 하는데 아무리 늦어도 $i = N$이 되는 순간에는 알고리즘이 종료될 것이기에 이 부분은 명확합니다.
- 알고리즘의 구조 상 소인수 목록에 적힌 수들의 곱이 $N$ 임은 자명합니다.
- 그리고 산술의 기본 정리에 의해 소인수 목록에 적힌 수들이 모두 소수이기만 하면 이 알고리즘이 올바른 소인수분해 결과를 준다는 것이 보장됩니다. 그러므로 소인수 목록에 적힌 수들이 모두 소수임을 증명해야 합니다.

# 0x00 소수 – 소인수분해


- 소인수 목록에 적힌 수들이 모두 소수라는 것을 증명해봅시다.
- 귀류법으로 소인수 목록에 합성수 a가 있다고 가정해봅시다. 그리고 a의 임의의 소인수 p를 생각해봅시다. 소인수 목록에 합성수 a가 있다는 것은 $i = a$일 때 당시 $N$이 a의 배수였다는 의미입니다. 그 말은 곧 $N$이 p의 배수라는 의미입니다.
- 그런데 $i = p$일 때 $N$에 있던 p의 소인수는 전부 제거가 되었습니다. 그러므로 $i = a$일 때 $N$은 p의 배수일 수 없기에 소인수 목록에 합성수 a가 있을 경우 모순이 발생합니다. 그러므로 명제가 증명되었습니다.

# 0x00 소수 – 소인수분해

- 어떻게 구현해야 할지 막막할 수도 있는데, 사실 구현은 굉장히 간단합니다.

```cpp
void solve(int n){
  for(int i = 2; n != 1; i++){
    while(n % i == 0){
      cout << i << '\n';
      n /= i;
    }
  }
}
```

# 0x00 소수 – 소인수분해


- 그런데 이 구현에도 최적화할 수 있는 여지가 있습니다. 최적화를 통해 최악의 경우 O($N$) 에서 O($\sqrt{N}$) 으로 줄일 수 있습니다.
- 이 알고리즘은 주어진 $N$의 소인수를 2부터 찾아나서는 방식입니다. 앞에서 소수 판정법의 시간복잡도를 줄이는 것과 같은 아이디어로 생각해보면 $i^2$ 가 $N$보다 커지면 그 즉시 $N$이 소수임을 알 수 있습니다.
- 이를 이용해 for문의 탈출 조건을 수정하면 $N$이 소수일 때 i 를 2부터 $N$-1까지 돌리는 대신 $i^2$ 가 $N$ 이하일 때 까지 돌리면 O($\sqrt{N}$) 으로 줄일 수 있습니다.
- 실제 구현할 땐 $N$ = 1이 되었을 때 1을 출력하지 않도록 주의해야 합니다.
- 예시 코드 : http://boj.kr/7a3f552f01c743d6b29298d76f7ea3be

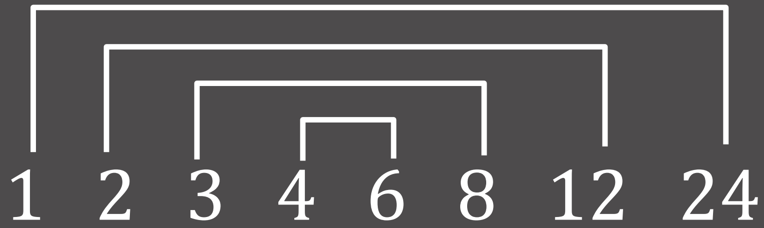
# 0x01 약수와 최대공약수

- 주어진 수 $N$의 약수 목록을 구해야 하는 문제를 생각해봅시다. (BOJ 2501번 : 약수 구하기)
- 가장 간단한 방법은 1부터 $N$까지 모든 수에 대해 $N$을 나누는지 확인하는 방법입니다.

```
vector<int> divisor(int n){
  vector<int> ret;
  for(int i = 1; i <= n; i++){
    if(n%i == 0) ret.push_back(i);
  }
}
```

# 0x01 약수와 최대공약수

- 그런데 약수 구하기 문제도 O($N$) 에서 O($\sqrt{N}$) 으로 줄일 수 있습니다. 바로 약수끼리 곱이 $N$이 되게끔 짝을 지을 수 있다는 성질을 이용하는 것입니다.
- 즉 $\sqrt{N}$ 이하의 약수만 구하고 나면 나머지 약수들은 $N$에서 그 약수들을 나눠 구할 수 있습니다. 단 $N$이 제곱수인 경우에 주의해야 합니다.




$N = 24$ 일 때


$N = 16$ 일 때($N$이 제곱수)

# 0x01 약수와 최대공약수

- 이 성질을 이용해 수정된 약수 목록을 반환하는 함수는 아래와 같습니다.

```
vector<int> divisor(int n){
  vector<int> ret;
  for(int i = 1; i*i <= n; i++){
    if(n%i == 0) ret.push_back(i);
  }
  for(int i = ret.size()-1; i >= 0; i--){
    if(ret[i]*ret[i] == n) continue;
    ret.push_back(n/ret[i]);
  }
}
```

# 0x01 약수와 최대공약수


- 최대공약수(Greatest Common Divisor)는 두 자연수의 공통된 약수 중 가장 큰 것을 의미합니다. GCD라고 줄여 쓰기도 합니다.(예 : GCD(6,20) = 2)
- 최소공배수(Least Common Multiple)은 두 자연수의 공통된 배수 중 가장 작은 것을 의미합니다. LCM라고 줄여 쓰기도 합니다.(예 : LCM(6,20) = 60)
- GCD를 구하기 위해 두 수 A, B의 약수 목록을 찾아 공통된 원소를 찾는 방법도 있지만 유클리드 호제법이라는 것을 사용하면 O(lg(max(A, B)))에 구할 수 있기에 더 효율적입니다.

**유클리드 호제법**

두 수 A, B에 대해 A를 B로 나눈 나머지를 r이라고 한다면 GCD(A, B) = GCD(B, r)이다.

- 예를 들어 56을 12로 나눈 나머지가 8이므로 GCD(56, 12) = GCD(12, 8)입니다.

# 0x01 약수와 최대공약수


- 재귀적으로 구현하면 아래와 같습니다. 왜 a와 b의 대소비교를 하지 않아도 잘 동작하는지 고민해보세요.

```
int gcd(int a, int b){
  if(a == 0) return b;
  return gcd(b%a, a);
}
```

- GCC에는 __gcd라는 아주 훌륭한 함수가 있으니 가져다쓰면 됩니다. (VS 2017에는 없습니다.) 해당 함수의 내부 구현은 위와 거의 동일합니다.(단 위와 달리 재귀 대신 반복문으로 구현이 되어있습니다.)

```
cout << __gcd(452,123);
```

# 0x01 약수와 최대공약수 – 몇 가지 성질들

- 성질 1. 두 수 A, B의 공약수들은 GCD(A, B)의 모든 약수들이다.
- 성질 2. 두 수 A, B의 공배수들은 LCM(A, B)의 모든 배수들이다.
- 성질 3. $A \times B = GCD(A, B) \times LCM(A, B)$
- 성질 4. $GCD(n, n+1) = 1$

- 성질 3을 이용해 LCM(A, B) 를 $A \times B \div GCD(A, B)$ 로 구할 수 있습니다. 성질 4는 보고 바로 까먹어도 아무 상관 없습니다.

# 0x02 합동식

- $A \equiv B \pmod{m}$ 이라는 기호의 의미는 A와 B가 M으로 나눈 나머지가 같다는 의미입니다.
- 깊게 파고들면 다룰 내용이 굉장히 많지만, 아주 얕은 내용만 소개해드릴 것입니다.
- $A \equiv B \pmod{m}$ 일 때

1. $A + C \equiv B + C \pmod{m}$
2. $A - C \equiv B - C \pmod{m}$
3. $AC \equiv BC \pmod{m}$ 입니다.
4. 그러나 $A \div C \equiv B \div C \pmod{m}$은 성립하지 않습니다. ( $A=6, B=2, C=2, M=4$ )

# 0x02 합동식


- 기호로 쓰니까 좀 낯설 수 있지만 사실 지금까지 문제를 풀 때 굉장히 자연스럽게 사용했던 성질들입니다.

### 출력

첫째 줄에 $1^K + 2^K + 3^K + \ldots + N^K$를 $\boxed{\text{1,000,000,007로 나눈 나머지를 출력한다.}}$

- 이런식으로 특정 값으로 나눈 나머지를 출력하는 문제에서 연산 중간과정에서 계속 나머지만을 챙긴 경험이 있을 것입니다.

```
ll POW(ll a, ll b, ll m){
  if(b==0) return 1;
  ll val = POW(a,b/2,m);
  val = val*val%m;
  if(b%2 == 0) return val;
  return val*a%m;
}
```

0x06강 – 재귀에서 거듭제곱을 수행하는 코드

# 0x02 합동식


- C언어로 특정 값으로 나눈 나머지를 출력하는 문제를 풀 때 int overflow 말고도 굉장히 조심해야 하는 부분은 음수의 나머지입니다.

```
cout << "result : " << -10 % 15;
```

```
result : -10
```

- A, B를 입력받아 A - B를 10으로 나눈 나머지를 출력하는 프로그램을 아래와 같이 짰다고 해봅시다. 이 코드는 잘 동작하는 것 같아 보입니다.

```
int mod = 10;
int main(void){
  int a,b;
  cin >> a >> b;
  int ans = (a-b)%mod;
  cout << "result : " << ans;
}
```

```
28626 51
result : 5
```

```
42 1
result : 1
```

```
5520 131
result : 9
```

# 0x02 합동식

- 그러나 A - B가 음수일 땐 잘못된 답을 출력합니다.

```
int mod = 10;
int main(void){
  int a,b;
  cin >> a >> b;
  int ans = (a-b)%mod;
  cout << "result : " << ans;
}
```

```
324 1351
result : -7
```

- 이를 방지하기 위해 아래와 같이 처리를 하는 것이 바람직합니다.

```
int mod = 10;
int main(void){
  int a,b;
  cin >> a >> b;
  int ans = (a-b)%mod;
  if(ans < 0) ans += mod;
  cout << "result : " << ans;
}
```

```
324 1351
result : 3
```

# 0x03 연립합동방정식

- BOJ 6064번 : 카잉 달력 문제를 풀어봅시다.
- 이 문제는 주어진 M, N, x, y 에 대해
- $A \equiv x \pmod{M}$, $A \equiv y \pmod{N}$ 인 A 를 찾는 문제입니다.
- 이런 연립합동방정식은 중국인의 나머지 정리(Chinese Remainder Theorem)을 사용하면 $O(\log M + \log N)$에 해결할 수 있습니다. 그러나 중국인의 나머지 정리를 이해하려면 모듈로 역수(Modular Inverse), 확장된 유클리드 호제법(Extended Euclidean Algorithm)을 먼저 알아야해서 중국인의 나머지 정리는 실전 알고리즘 강의에서 다루지 않을 예정입니다.
- 대신 시간복잡도는 조금 나쁘지만 이해하기 쉬운 풀이법을 설명드릴 예정입니다.

# 0x03 연립합동방정식


$A \equiv x \pmod{M}$, $A \equiv y \pmod{N}$ 인 A 를 출력해라. 존재하지 않는다면 -1을 출력해라.

- 성질 1. A가 존재한다면 1~ LCM(M, N) 사이에 유일하게 존재합니다.
  이 성질은 귀류법을 통해 증명할 수 있습니다만 생략하겠습니다.

- 성질 2. A가 존재할 필요충분조건은 $x \equiv y \pmod{\mathrm{GCD}(M, N)}$ 이다.
  이 성질 또한 귀류법으로 양방향을 다 보임으로서 증명할 수 있지만 생략하겠습니다.

# 0x03 연립합동방정식


$A \equiv x \pmod{M}$, $A \equiv y \pmod{N}$ 인 A 를 출력해라. 존재하지 않는다면 -1을 출력해라.

- 성질 1을 이용해서 A에 1부터 LCM(M, N)까지 차례대로 넣어봄으로서 답을 구할 수 있습니다. LCM(M, N) 을 계산하기 싫으면 $\mathrm{LCM}(M, N) \le MN$ 이므로 그냥 1부터 MN까지 계산해도 결과는 동일하게 나옵니다. 만약 만족하는 A가 하나도 없으면 -1을 출력하면 됩니다.

```
int solve(int m, int n, int x, int y){
  if(x == m) x = 0;
  if(y == n) y = 0;
  for(int i = 1; i <= m*n; i++){
    if(i % m == x and i % n == y) return i;
  }
  return -1;
}
```

# 0x03 연립합동방정식


$A \equiv x \pmod{M}$, $A \equiv y \pmod{N}$ 인 A 를 출력해라. 존재하지 않는다면 -1을 출력해라.

- 그러나 이 방식은 O(MN)이기 때문에 MN이 최대 1,600,000,000인 이 문제에서는 시간 초과가 발생합니다. O(MN) 대신 O(N)으로 개선하고 싶습니다.
- i를 1부터 MN까지 하나하나 다 해보는 대신 조금 더 효율적인 방법이 있지 않을까요?
- $A \equiv x \pmod{M}$이기 때문에 A가 존재한다면 $x, x+M, x+2M, x+3M, \cdots$ 중에 하나입니다. 그렇기에 i를 $x, x+M, x+2M, x+3M, \cdots$ 에 대해서만 해보면 되겠네요.
- 예시 코드 : http://boj.kr/ffb4bb162d9f4dfba2cae83895badda5

# 0x03 연립합동방정식

$A \equiv x \pmod M$, $A \equiv y \pmod N$, $A \equiv z \pmod K$ 인 A 를 출력해라. 존재하지 않는다면 -1을 출력해라.

- 이 문제를 푸는 시간복잡도를 O(MNK)에서 어디까지 떨굴 수 있을까요?
- 물론 중국인의 나머지 정리를 쓰면 log scale로 떨굴 수 있지만, 앞의 문제와 같이 간단한 방법으로도 O(M+N)까진 거뜬합니다. 한 번 고민해보세요.

# 0x04 이항계수와 팩토리얼

- 순열과 조합을 모르면 이번 장을 듣기 전에 먼저 고등학교 수학 수준의 순열과 조합 지식을 공부하고 오시는걸 추천드립니다.
- $n\,C\,k$ 를 구하는 문제를 생각해봅시다. 첫 번째는 $n$과 $k$가 10이하인 문제입니다.(BOJ 11050번 : 이항 계수 1)
- $n\,C\,k = n!\,/\,(n-k)!k!$ 이라는 식을 통해 쉽게 답을 얻을 수 있습니다.
- 예시 코드 : http://boj.kr/88189dab63974f8e9446dd9d960aea56

# 0x04 이항계수와 팩토리얼

- 두 번째는 n과 k가 1000이하인 문제입니다.(BOJ 11051번 : 이항 계수 2)
- n C k = n! / (n-k)!k! 이라는 식을 이용하고 싶지만, 최대 1000! 을 `int`나 `long long`에 담을 수 없음은 자명합니다. `double`에 담더라도 실수오차로 인해 정확한 값이 담길 리가 없습니다.
- n!, (n-k)!, k! 을 10,007로 나눈 나머지는 알 수 있습니다. 그러나 우리는 현재 합동식 안에서 덧셈, 뺄셈, 곱셈은 수행할 수 있지만 나눗셈을 수행하는 법을 모릅니다.
- 이 문제는 n C k = n-1 C k + n-1 C k-1 이라는 식을 통해 다이나믹 프로그래밍으로 해결해야 합니다. int overflow에 주의하세요.
- 예시 코드 : http://boj.kr/a83ad6289e2448caa16a643b2858047f

# 0x04 이항계수와 팩토리얼

- BOJ 1676번 : 팩토리얼 0의 갯수 문제를 풀어봅시다.
- N의 뒤에서부터 처음 0이 아닌 숫자가 $k$개 나온다는 것은 곧 N이 $10^k$ 의 배수이면서 $10^{k+1}$ 의 배수가 아니라는 의미입니다.
- 그렇기에 N!을 소인수분해 했을 때 $N! = 2^a \times 5^b \times etc$ 라면 N!의 뒤에서부터 처음 0이 아닌 숫자가 나올 때 까지 0이 $\min(a, b)$개 나옵니다.
- 상식적으로 생각해서 N! 에 2보다 5가 많을테니 5가 몇 개인지만 세면 됩니다. N이 최대 500이므로 1~ N 에서 5의 배수의 갯수, 25의 배수의 갯수, 125의 배수의 갯수를 더하면 됩니다.
- 예시 코드 : http://boj.kr/1f2d4999b1804d1ca1e8c65354de0e32

# 0x04 이항계수와 팩토리얼


- 아래의 코드는 또 다른 정답 코드입니다. 어떤 원리로 성립하는 것일지 한 번 고민해보세요.

```
int main() {
  ios::sync_with_stdio(0);
  cin.tie(0);
  int n;
  cin >> n;
  int cnt = 0;
  while(n > 0){
    n /= 5;
    cnt += n;
  }
  cout << cnt;
}
```

# 강의 정리


- 소수 판정법, 에라토스테네스의 체, 소인수분해를 익혔습니다. 에라토스테네스의 체의 최적화도 그닥 어려운 내용이 아니니 꼭 익혀두세요.
- 약수를 구하는 방법과 더불어 유클리드 호제법, GCD와 LCM에 관련된 몇 개의 성질을 익혔습니다.
- 합동식의 성질, 나머지를 구할 때 주의사항, 연립합동방정식을 푸는 방법을 익혔습니다.
- 이항계수를 구하는 방법과 팩토리얼과 관련한 문제를 풀어보았습니다.