최대공약수 & 최소공배수

① 최대공약수 (GCD, Great Common Divisor) : 유클리드 호제법으로 구한다.

* 유클리드 호제법의 기본 원리

A>B 라고 가정하고 두 수 A, B의 GCD를 구한다고 했을 때,

A와 B의 최대공약수 = A-B와 B의 최대공약수 ∵ 이를 반복

= A%B와 B의 최대공약수

이다. 즉, GCD(A, B) = GCD(A-B, B) = GCD(A-2B, B) = ⋯ = GCD(A%B, B)

ex) GCD(28, 8) = GCD(20, 8) = GCD(12, 8) = GCD(4, 8)

→ 28 ÷ 8 = 3 … 4

* 유클리드 호제법 과정

```
int gcdIter(int a, int b) {
    while (b) { // b == 0, a가 최대 공약수
        a %= b;     // a = a % b;
        swap(a, b);
    }
    return a;
}
```

➤ 출처 : 양들비튜고 강의자료 中 정우준 자료. (본인 작성 코드X)

① 두 정수 a, b가 주어진다. (a > b 라고 가정)

② a%b를 해준다

③ a%b를 하면 a<b가 되므로 swap(a, b)

④ a%b가 0이 된다면 최대공약수를 찾은 것이므로

반복문을 빠져나오도록 b==0 조건을 걸어주고

최대공약수인 a를 return 한다

② 최소공배수 (LCD, Least Common Multiple)

" 두 수의 곱 = 최대공약수 × 최소공배수 " 식을 이용해서 구한다.

∵ 최소공배수 = 두 수의 곱 ÷ 최대공약수

소수 (약수가 1과 자기자신 뿐인 수)

소수는 에라토스테네스의 체를 이용하여 구한다.

* 에라토스테네스의 체를 이해하기 위한 기본지식

2 ~ n까지 돌아면서 나눠지는 수가 없으면 소수로 판별 ↔ 더 간단하게 2~$\sqrt{n}$ 까지만 돌려도 된다!

∵ $a = n / (\sqrt{n}$보다 같거나 큰 수) 라는 a가 존재하면, a는 2이상 $\sqrt{n}$ 이하다.

따라서 $\sqrt{n}$까지만 확인하면 그 이상은 확인할 필요X.

↔ ex) n=8 , $8 = \underbrace{2 \times 4}_{\text{대칭}} = 4 \times 2$

* 에라토스테네스의 체

```
                    ↱ 모든 index가 true로 되어있는 vector
void isPrime(int n, vector<bool> &is_prime) {
    //0과 1은 소수가 아니므로 먼저 제거
    is_prime[0] = is_prime[1] = false;
    //2 ~ 제곱근 n까지 검사
    for (int i = 2; i <= sqrt(n); i++) {
        if (is_prime[i]) { //i가 소수라면 (true로 되어있다면)
            for (int j = i * i; j <= n; j += i) {
                is_prime[j] = false; //i의 배수를 제거
            }
        }
    }
}
```

▶ 출처 : 알고비튼2 강의자료 中 정우준 자료. (본인 작성 코드X)

① 맨처음에 모든 인덱스가 true로 되어있는 "is_prime" vector는 소수가 아닌 인덱스의 값을 false로 바꿔주어서, 최종적으로는 소수인 인덱스 값만 true로 남게될 것이다.

② 0, 1은 소수가 아니므로 바로 false를 넣어준다.

③ for문으로 2부터 시작해서 해당 숫자의 배수에 해당하는 숫자들을 지워나간다. (두번째 반복문)

∵ 어떤 수의 배수가 되는 수는 당연히 소수가 아니다!

이때 j = i * i 부터 시작하는 이유는, i-1 까지는 앞에서 이미 지워졌기 때문이다.

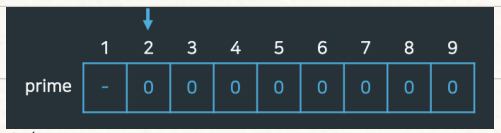
ex) 5의 배수 中 10, 20, 30, … 과 같은 짝수는 2의 배수에서 이미 지운 상태

④ $\sqrt{n}$까지 갔는데도 아직 true가 남아있다면 그것들은 어차피 다 소수 ∴ 숫자(n)까지 반복문을 돌린다(첫번째 반복문)

어려운 소인수분해 - 소수 구하기 응용

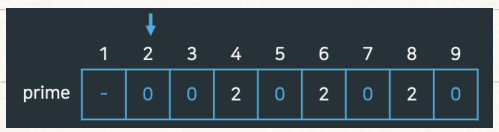
- 소인수분해를 소수를 구하는 에라토스테네스의 체를 활용하여 구할 수 있다.

- 소인수분해 하기



▶ 출처 : 양들비튜고 강의자료 中 정수론 자료. (본인 작성 코드X)

↳ 위와 같이 True / False 가 아니라 숫자 0으로 저장되어 있다. 값을 모두 업데이트하고도 0이라면 그 수는 소수!



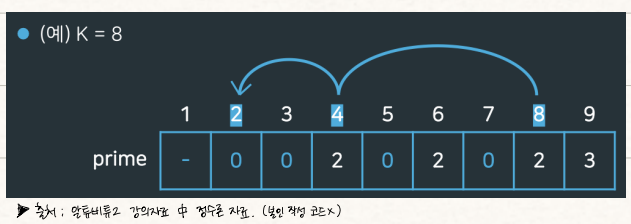
▶ 출처 : 양들비튜고 강의자료 中 정수론 자료. (본인 작성 코드X)

↳ 2는 소수이고, 2의 배수에 접근하는데 이때 해당 인덱스에는 '소수 판단 여부 정보'가 아니라 '어느 소수의 배수' 인지 저장한다.

즉 2의 배수인 4, 6, 8 … 번 인덱스에는 2가 저장되을 것이다.

↳ 이런 식으로 반복하는데, 이미 값이 존재한다면 갱신하지 않는다.

예를 들어 2의 배수 中 10번 인덱스에 2가 이미 저장되어있다면, 5의 배수 조사시 10번 인덱스에 5를 갱신하지 않는다는 것!



▶ 출처 : 양들비튜고 강의자료 中 정수론 자료. (본인 작성 코드X)

↳ 모두 완료했다면, 이제 역으로 경로를 조사한다.

✓ 8 -> prime[8] = 2 (출력) -> 8/2 = 4 → 8번 인덱스에 2 들어가 있으므로 2 출력 후, 8/2 = 4
✓ 4 -> prime[4] = 2 (출력) -> 4/2 = 2 → 4번 인덱스에 2 들어가 있으므로 2 출력 후, 4/2 = 2
✓ 2 -> prime[2] = 0 -> 남은 2 출력 후 종료 → 2번 인덱스에 0 들어가 있으므로 2 출력 후 종료!