수학

목차

- 나머지 연산
- 최대공약수(GCD)/최소공배수(LCM)
- 소수
  - 소수 구하기.
  - 에라토스테네스의 체

나머지 연산

문제를 풀다 보면 답이 매우 큰 경우가 있다. 이럴 때 문제에서 '정답을 n으로 나눈 나머지를 구하여라 ' 라고 주어지는 경우가 있다.

- $(A + B)\ mod\ M\ =\ ((A\ mod\ M\ )\ +\ (B\ mod\ M))\ mod\ M$
- $(A * B)\ mod\ M\ =\ ((A\ modM) * (BmodM))modM$
- 나누기는 성립 x
- $(A - B)mod\ M\ =\ (\ (\ A\ mod\ M) - (B\ mod\ M)\ +\ M)\ mod\ M$(음수 보정)

나머지 연산(음수보정..?)

- (6%3-5%3)%3 을 풀어보자.

C11,C++14: -2
Java : -2
Python3: 1

언어마다 다르다.
그래서 음수를 보정을 해 주는 것. (원하는 결과를 얻기 위해)

부등호로 증명하지만 필자는 이해를 못했다.

시간복잡도

- https://joshuajangblog.wordpress.com/2016/09/21/time_complexity_big_o_in_easy_explanation/

최대공약수

- 최대공약수 (Greatest Common Divisor) 즉, GCD라고 부른다.
- 두 수 A,B의 공통된 약수 중에서 가장 큰 수
- 최대공약수가 1인 두 수를 서로소(Coprime)라고 한다.
- 최대공약수를 가장 쉽게 구하는 방법은 2부터 min(a,b)까지 나누어 보면 된다.

```cpp
#include <iostream>

int min(const int &a, const int &b) {
    return a < b ? a : b;
}

int main(void) {
    int result;
    int a, b;

    for (int i = 2; i <= min(a, b); i++) {
        if (a & i == 0 && b % i == 0)
            result = i;
    }
    return 0;
}
```

최대공약수

- 앞에 있는 방법은 시간 복잡도가O(n)이다.
  - ‘최악의 시간이 $\min(a,b)$만큼 돌기 때문에. (여기선 $\min(a,b)$돈다.)
- O(n)도 좋은 알고리즘이지만 더 좋은 알고리즘이 있다.

최대공약수-유클리드 호제법

- A를 b로 나눈 나머지를 r이라고 했을 때
- GCD(a,b) = GCD(b,r)이다.
- R이 0이면 그때 b가 최대 공약수이다.
- GCD(24,16) = GCD(16,8) = GCD(8,0) = 8
- 이 내용을 이용하여 gcd를 구하는 함수를 구해보자

최대공약수-유클리드 호제법

```c
int main(void) {
    int a, b;
    while (b != 0) {
        int r = a % b;
        a = b;
        b = r;
    }
    //GCD(a,b) = 여기서의 b의 값.
    return 0;
}
```

While버전

```c
#include <iostream>

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

재귀버전

이 시간 복잡도는 $O(logN)$이라고 한다

최대공약수-유클리드 호제법

GCD(a,b,c) = GCD(GCD(a,b),c)
계속 늘려가도 성립된다.

최소공배수

1. Least Common Multiple 줄여서 LCM이라고 한다.
2. 두 수의 공통된 배수 중에서 가장 작은 수.
3. GCD를 응용해서 구할 수 있다.
4. 두 수 a,b의 최대공약수를 g라고 했을 때,
5. 최소공배수 l 은 g*(a/g)*(b/g) 이다.

두 정수의 최대공약수와 최소공배수의 관계는 다음과 같다.

$$\gcd\{n,m\}\operatorname{lcm}\{n,m\} = nm$$

https://ko.wikipedia.org/wiki/%EC%B5%9C%EB%8C%80%EA%B3%B5%EC%95%BD%EC%88%98

두수 곱한 거에 gcd 나눠주면 최소공배수다.

최소공배수

같이 만들어 본 gcd와 lcm을 이용하여 문제를 풀어보자

소수

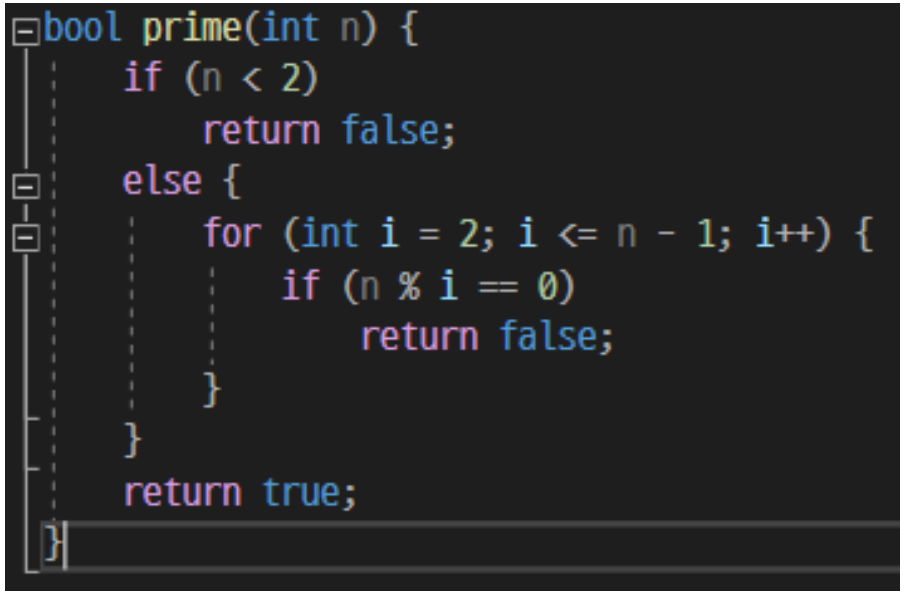
- 약수가 1과 자기 자신 밖에 없는 수.
- 어떤 수가 소수이기 위해선, 2보다 크고 N-1보다 작거나 같은 자연수로 나누어 떨어지면 안된다.
- 어떤 수 N이 소수인가 아닌가.
- N이하의 소수를 모두 찾기.

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

간단하지만 이 방법도 O(n)이다.

소수

어떤 수가 소수이기 위해선, 2보다 크거나 같고, 루트N보다 작거나 같은
자연수로 나누어 떨어지면 안된다.
N이 소수가 아니라면, N = a*b 로 나타낼 수 있다. (a <= b)
두 수 a와 b의 차이가 가장 작은 경우는 루트 N이다. (루트의 성질)
따라서 루트 N까지만 검사해보면 된다.
24 = 1, 2, 3, 4, 6, 8, 12, 24



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

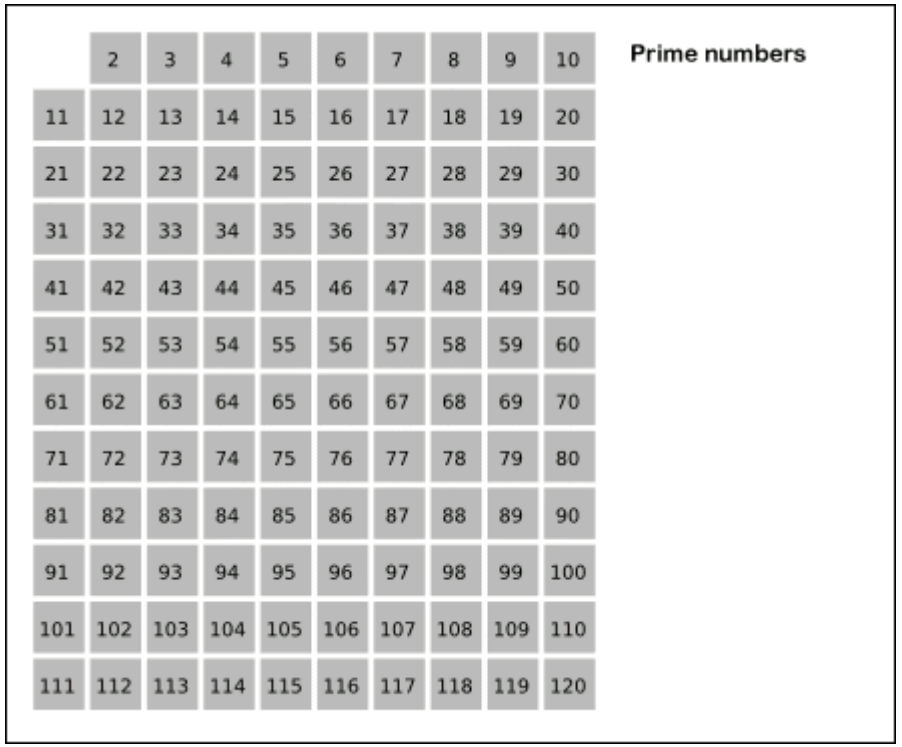
루트 n은 근사값이기 때문에
i<=n-1을 i*i<=n으로 변경하자!
이러면 O(루트n)으로 훨씬 빨라졌다.

그럼 1~n까지 소수의 개수를 찾는 문제의 시간복잡도는?

소수

그럼 1~n까지 소수의 개수를 찾는 문제의 시간 복잡도는 O(n$\sqrt{n}$)이다.
좋아 보이지만 1,000,000까지의 소수를 구하려면 1,000,000 * 1,000 만큼 걸린다.
10억 = 10초

소수



에라토스테네스의 체라고 한다.
시간 복잡도가 무려 O(n$loglogn$).
루트N -> $loglogn$ 이면 아까 1,000,000의 수를 본다면,
N * 1,000 -> N * $log3$ 으로 줄어드는 것이다.

한번 오른쪽의 그림을 보고
구현해보는 시간을 가져보자.

함수호출시, 파라메터에 N을 넣으면 N까지의 소수를
구해주는 함수를 만들어보자.

소수

```cpp
const int ERA_MAX = 1000000;    //1,000,000까지의 수를 담을 예정
bool check[ERA_MAX];    //소수가 아닌지 체크
vector<int> v;
void go_era(int num) {
    if(num <= 1 || num > 1000000)
        return;
    else {
        for(int i=2;i*i<=num;i++) { //N이 소수인지 판별할때 sqrt(N)까지 체크
            if(!check[i]) { //체크가 되지 않은 수이면
                v.push_back(i); //넣는다
                for(int j=i+i;j<=num;j*=2) {
                    check[j] = true;  //체크
                }
            }
        }
    }
}
```

f go_era

대충 비슷한가?

소수

1929 소수 구하기