

#2. 정수론 (1)

나정휘

<https://justicehui.github.io/>

목차

소수 판별

소인수분해

유클리드 호제법

에라토스테네스의 체

곱셈적 함수

용어 정의

소수

- 약수가 1과 자기 자신밖에 없는 2 이상의 자연수
- 소수는 무한히 많음
 - 소수가 유한하다고 가정하고 귀류법 사용하면 증명 가능
- 임의의 자연수 n 에 대해, 소수가 등장하지 않는 길이 n 인 구간 존재
 - $2 \leq k \leq n+1$ 일 때 $(n+1)! + k$ 는 k 의 배수이므로 소수가 아님
- 임의의 자연수 n 에 대해, $n < p \leq 2n$ 인 소수 p 가 존재
 - 베르트랑 공준
- $\pi(x)$ 를 x 이하 소수의 개수라고 하면, $\lim_{n \rightarrow \infty} \frac{\pi(x) \log x}{x} = 1$ 이 성립함
 - x 이하 소수의 개수는 $O(\frac{x}{\log x})$ 개
 - 소수 정리

용어 정의

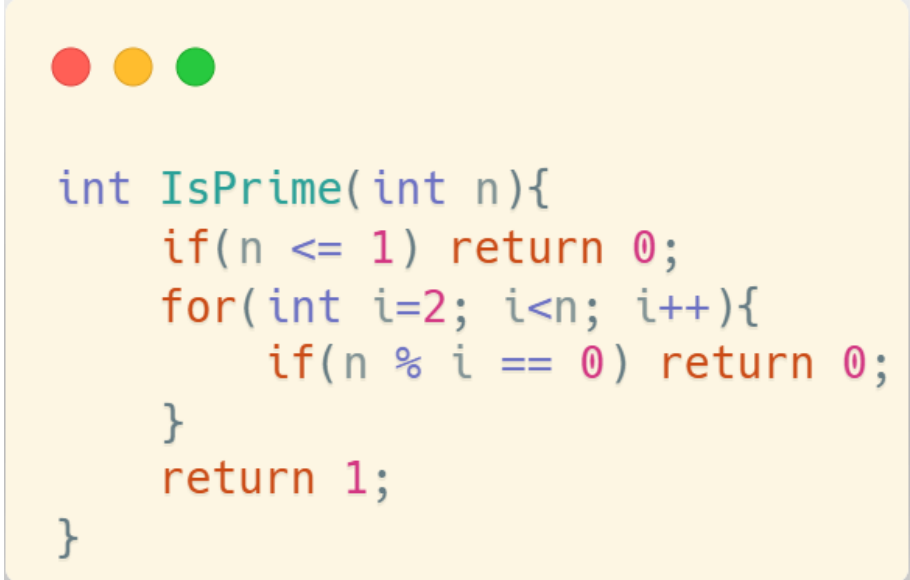
2 이상의 자연수 n 을 소수들의 곱으로 표현하는 것: 소인수분해

- $n = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$
- 소수들의 순서만 다른 경우를 같은 표현으로 보면, 소인수분해의 결과는 유일하게 존재
 - 2개 이상이라고 가정하고 귀류법 사용하면 증명 가능
 - 산술의 기본 정리
- n 의 약수는 $p_1^{f_1} p_2^{f_2} \dots p_k^{f_k}$ 꼴 (단, $0 \leq f_i \leq e_i$)

소수 판별

목표: 양수 n 이 주어졌을 때 n 이 소수이면 1, 소수가 아니면 0 반환하는 함수 작성

- $n = 1$ 이면 0 반환
- 소수의 약수는 1과 자기 자신밖에 없으므로 $2, 3, \dots, n - 1$ 중 하나로 나누어 떨어지면 0 반환
- 필요한 나눗셈 연산의 횟수: n 번



```
int IsPrime(int n){  
    if(n <= 1) return 0;  
    for(int i=2; i<n; i++){  
        if(n % i == 0) return 0;  
    }  
    return 1;  
}
```

소수 판별

목표: 양수 n 이 주어졌을 때 n 이 소수이면 1, 소수가 아니면 0 반환하는 함수 작성

- 사실 $\sqrt{n}$ 이하까지만 확인해도 됨
 - $n = pq$ ($p \leq q$)일 때 $p \leq \sqrt{n}$ 을 만족함
 - $p, q > \sqrt{n}$ 이면 $n < pq$
 - 동일한 방식으로 n 의 모든 약수를 찾을 수 있음
- 필요한 나눗셈 연산의 횟수: $\sqrt{n}$ 번

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

소수 판별

목표: 양수 n 이 주어졌을 때 n 이 소수이면 1, 소수가 아니면 0 반환하는 함수 작성

- 연산을 조금 더 줄이고 싶다면...
 - 2를 제외한 모든 소수는 홀수
 - 따라서 $n = 2$ 인 경우를 제외하면 홀수 i 만 봐도 충분함
 - 필요한 나눗셈 연산의 횟수: $\sqrt{n}/2$ 번

```
int IsPrime(int n){  
    if(n == 2) return 1;  
    if(n <= 1 || n % 2 == 0) return 0;  
    for(int i=3; i*i<=n; i+=2){  
        if(n % i == 0) return 0;  
    }  
    return 1;  
}
```

소수 판별

목표: 양수 n 이 주어졌을 때 n 이 소수이면 1, 소수가 아니면 0 반환하는 함수 작성

- 연산을 조금 더 줄이고 싶다면...
 - 2를 제외한 모든 소수는 홀수
 - 따라서 $n = 2$ 인 경우를 제외하면 홀수 i 만 봐도 충분함
 - 필요한 나눗셈 연산의 횟수: $\sqrt{n}/2$ 번
- 3보다 큰 모든 소수는 $6k+1$ 또는 $6k+5$ (or $6k-1$)꼴
 - $6k, 6k+2, 6k+4$ 는 2의 배수
 - $6k+3$ 은 3의 배수
- 필요한 나눗셈 연산의 횟수: $\sqrt{n}/3$ 번

```
int IsPrime(int n){
    if(n == 2 || n == 3) return 1;
    if(n <= 1 || n % 2 == 0 || n % 3 == 0) return 0;
    for(int k=1; (6*k-1)*(6*k-1)<=n; k++){
        if(n % (6*k-1) == 0) return 0;
        if(n % (6*k+1) == 0) return 0;
    }
    return 1;
}
```

소인수분해

목표: 양수 n 의 소인수를 모두 출력하는 것

- 가장 단순한 방법
 - 2부터 n 까지의 모든 정수를 차례대로 보면서
 - n 을 나눌 수 있으면 계속 나누기
- 필요한 나눗셈 연산의 횟수: 최대 $n + \log_2 n$ 번
 - n 이 어떤 정수 $2 \leq k \leq n$ 으로 나뉘는지 확인하는 부분에서 $n - 1$ 번
 - 실제로 n 을 나누는 것은 최대 $\log_2 n$ 번
 - 나눌 때마다 절반 이하가 되고, 1이 되면 더 이상 나눌 수 없음



```
void Factorize(int n){
    for(int i=2; i<=n; i++){
        while(n % i == 0){
            cout << i << "\n";
            n /= i;
        }
    }
}
```

소인수분해

목표: 양수 n 의 소인수를 모두 출력하는 것

- n 은 $\sqrt{n}$ 보다 큰 소인수를 중복을 포함해 최대 한 개 가질 수 있음
- $\sqrt{n}$ 이하의 소수로 모두 나눠보면 됨
- 만약 마지막에 $n \neq 1$ 이라면 n 도 소인수
- 필요한 나눗셈 연산의 횟수: 최대 $\sqrt{n} + \log_2 n$ 번
- $\sqrt{n}$ 이하의 소수만 사용하면 조금 더 빨라짐

```
void Factorize(int n){  
    for(auto i : Primes){  
        if(i * i > n) break;  
        while(n % i == 0){  
            cout << i << " ";  
            n /= i;  
        }  
    }  
    if(n != 1) cout << n << " ";  
    cout << "\n";  
}
```

유클리드 호제법

최대공약수의 성질

- $\gcd(a, b) = \gcd(|a|, |b|)$
- $\gcd(a, 0) = |a|$
- $\gcd(a, b) = \gcd(b, a)$
- $\gcd(a, b) = \gcd(a \pm b, b)$
 - $d|a \text{ and } d|b \Leftrightarrow d|(a + b) \text{ and } d|b$
 - 이므로 (a, b) 의 공약수 집합과 $(a+b, b)$ 의 공약수 집합 동일함
 - $a-b$ 도 동일하게 증명 가능
- $\gcd(a, b) = \gcd(a + nb, b)$
 - 위의 결과에서 수학적 귀납법 적용
- $\gcd(a, b) = \gcd(a \bmod b, b)$
 - $a \bmod b = r$ 이라고 하면 $a = nb + r$ 인 정수 n 존재
 - a 에서 b 를 여러 번 뺀다고 생각해도 됨

유클리드 호제법

유클리드 호제법

- 두 정수 a, b 의 최대공약수를 구하는 과정
 - 음수인 경우 절댓값을 취하면 되므로 $a, b \geq 0$ 인 경우만 생각
 - $\gcd(a, b) = \gcd(b, a)$ 이므로 $a \geq b$ 인 경우만 생각
 - $\gcd(a, 0) = a$ 이므로 $a \geq b \geq 1$ 인 경우만 생각, $b = 0$ 이면 알고리즘 종료
- $\gcd(a, b) = \gcd(a \bmod b, b)$
 - $a = bq + r$ 이라고 하면 $\gcd(a, b) = \gcd(b, r)$, $a \geq b > r$
 - (a, b) 를 (b, r) 로 축소
 - 이대로 b 를 0까지 끌고 내려가면 됨
 - $q \geq 1$ 이므로 $2r \leq (q + 1)r = qr + r \leq qb + r = a$
 - 따라서 $r \leq a/2$ 이므로 $br \leq ab/2$
 - $O(\log ab)$ 번의 축소를 거치면 $b = 0$

유클리드 호제법

BOJ 2609. 최대공약수와 최소공배수

- 두 자연수의 최대공약수와 최소공배수를 출력하는 문제
- $\text{lcm}(a, b) = a * b / \text{gcd}(a, b)$
 - 계산 과정에서 나올 수 있는 최댓값은 ab
- $\text{lcm}(a, b) = a / \text{gcd}(a, b) * b$
 - 계산 과정에서 나올 수 있는 최댓값은 $\text{lcm} \leq ab$



```
int gcd(int a, int b){  
    return b ? gcd(b, a % b) : a;  
}  
  
int lcm(int a, int b){  
    return a / gcd(a, b) * b;  
}
```

에라토스테네스의 체

목표: 1부터 n 까지의 소수를 모두 구하는 것

- 2는 소수, 2의 배수를 전부 제거
- 3은 소수, 3의 배수를 전부 제거
- 4는 이미 제거됨
- 5는 소수, 5의 배수를 전부 제거
- 6은 이미 제거됨
- 7은 소수, 7의 배수를 전부 제거
- ...

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

에라토스테네스의 체

반복문의 반복 횟수

- n 이하의 소수 p 에 대해, n 보다 작거나 같은 p 의 배수를 제거
- $\sum_{p \leq n} \frac{n}{p} = n \sum_{p \leq n} \frac{1}{p} \leq n \sum_{k=1}^n \frac{1}{k} \approx n \ln n$
- 따라서 약 $n \ln n$ 번 이하의 반복만 필요함

- 사실 $\sum_{p \leq n} \frac{1}{p} \approx \ln \ln n$ 이라서 실제로는 약 $n \ln \ln n$ 번 이하의 반복만 필요함
 - Mertens' second theorem

에라토스테네스의 체

BOJ 15965 K번째 소수

- $K \leq 500'000$ 번째 소수를 출력하는 문제
- 소수 정리에 의해 K번째 소수는 약 $K \ln K$ 정도
 - 800만 이하에서 나옴
- 사실 j는 $i*i$ 부터 시작해도 됨
 - $2i, 3i, 4i, \dots, (i-1)i$ 는 이미 앞에서 지워짐

```
#include <bits/stdc++.h>
using namespace std;

int Check[8080808];
vector<int> Primes;

void Sieve(int n){
    for(int i=2; i<=n; i++){
        if(Check[i]) continue;
        Primes.push_back(i);
        for(int j=i+i; j<=n; j+=i) Check[j] = 1;
    }
}

int main(){
    Sieve(8'000'000);
    int K; cin >> K;
    cout << Primes[K-1];
}
```

에라토스테네스의 체

에라토스테네스의 체 다시 살펴보기

- 소수 p 의 배수를 지우는 과정을 다시 생각해 보자.
- 만약 x 가 p 에 의해서 처음으로 지워졌으면
- p 는 x 의 가장 작은 소인수
- 가장 작은 소인수가 곱해진 횟수 $e[i]$ 도 알 수 있음
 - $sp[j/i] = i$ 이면 $e[i] = e[j/i] + 1$
 - 그렇지 않으면 $e[i] = 1$

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

```
int SP[1010101], E[1010101];

void Sieve(int n=1'000'000){
    for(int i=2; i<=n; i++){
        if(SP[i]) continue;
        for(int j=i; j<=n; j+=i){
            if(SP[j]) continue;
            SP[j] = i;
            E[j] = SP[j/i] == i ? E[j/i] + 1 : 1;
        }
    }
}
```

에라토스테네스의 체

소인수분해

- x 의 가장 작은 소인수를 $sp[x]$ 라고 하면
- $\text{while}(x > 1) x /= sp[x];$
- 최대 $\log_2 x$ 번의 나눗셈으로 x 소인수분해 가능
 - x 는 매번 절반 이상 감소

```
#include <bits/stdc++.h>
using namespace std;

int SP[5050505];
void Sieve(int n){
    for(int i=2; i<=n; i++){
        if(SP[i]) continue;
        SP[i] = i;
        for(int j=i+i; j<=n; j+=i) if(!SP[j]) SP[j] = i;
    }
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    Sieve(5000000);
    int N; cin >> N;
    for(int i=1; i<=N; i++){
        int X; cin >> X;
        while(X > 1){
            cout << SP[X] << " ";
            X /= SP[X];
        }
        cout << "\n";
    }
}
```

곱셈적 함수

곱셈적 함수

- 서로소인 두 자연수 n, m 에 대해 $f(n)f(m) = f(nm)$ 이 성립하는 함수 f 를 곱셈적 함수라고 부름
- 소인수분해
 - 소수의 거듭제곱에 대한 함숫값을 빠르게 구할 수 있으면
 - 소인수분해를 이용해 임의의 자연수의 함숫값을 빠르게 구할 수 있음
 - 편의상 n 를 소인수분해해서 $n = \prod p_i^{e_i}$ 로 나타내자.
 - $f(n) = \prod f(p_i^{e_i})$
- 예시
 - 양의 약수의 개수 $\tau(n) = \prod (e_i + 1)$
 - 양의 약수의 합 $\sigma(n) = \prod (1 + p_i + p_i^2 + \cdots + p_i^{e_i}) = \prod \frac{p_i^{e_i+1} - 1}{p_i - 1}$
 - n 이하 자연수 중 n 과 서로소인 개수 $\phi(n) = n \prod (1 - 1/p_i)$
 - 이걸 지금 몰라도 됨

곱셈적 함수

곱셈적 함수의 계산

- 에라토스테네스의 체를 이용하면 $f(1), f(2), \dots, f(n)$ 를 모두 계산할 수 있음
- n 의 최소 소인수를 p_k 라고 하면
$$f(n) = f\left(\frac{n}{p_k^{e_k}}\right) f(p_k^{e_k}) = \frac{f\left(\frac{n}{p_k}\right) f(p_k^{e_k})}{f(p_k^{e_k-1})}$$

```
int SP[1010101], E[1010101], Tau[1010101];
ll Sigma[1010101];

int GetTau(int p, int e){ return e + 1; }
ll GetSigma(int p, int e){ return (Pow(p, e+1) - 1) / (p - 1); }

void Sieve(int n=1'000'000){
    for(int i=2; i<=n; i++){
        if(SP[i]) continue;
        for(int j=i; j<=n; j+=i){
            if(SP[j]) continue;
            SP[j] = i; E[j] = SP[j/i] == i ? E[j/i] + 1 : 1;
        }
    }
    Tau[1] = Sigma[1] = 1;
    for(int i=2; i<=n; i++){
        Tau[i] = Tau[i/SP[i]] / GetTau(SP[i], E[i]-1) * GetTau(SP[i], E[i]);
        Sigma[i] = Sigma[i/Pow(SP[i], E[i])] * GetSigma(SP[i], E[i]);
    }
}
```