김범수

수학

math

By POSCAT

Contents

소수 판별 03

유클리드 알고리즘 07

확장된 유클리드 알고리즘 09

소수 판별

- 어떤 수 n에 대해, n이 1과 자기 자신으로만 나누어 떨어진다면 우리는 n을 소수라고 합니다.
- 그렇다면 n이 소수인지 아닌지는 어떻게 알 수 있을까요?
- 단순히 2부터 n – 1까지 모든 수를 n으로 나눠보면서 만약 나누어 떨어지는 경우가 존재하면 n이 소수가 아니란 것을 알 수 있습니다.
- 하지만 n – 1까지 모든 수로 나눌 필요가 있을까요?
- 조금만 생각해보면 $\sqrt{n}$까지만 나눠봐도 n이 소수인 지를 알 수 있습니다.

소수 판별

- n이 소수인 지를 알기 위해서는 $O(\sqrt{n})$의 시간이 필요합니다. 하지만 n이하의 모든 소수를 알고 싶다면 어떻게 해야 할까요? 2부터 n까지 모든 수에 대해 일일이 소수인 지를 판별하면 $O(n\sqrt{n})$의 시간이 걸립니다. 이는 너무 오래 걸립니다.
- 에라토스테네스의 체를 이용하면 시간복잡도를 줄일 수 있습니다. 에라토스테네스는 아래의 과정으로 이뤄집니다.

  1. 2부터 n까지 적는다.
  2. 2를 제외한 모든 2의 배수를 지운다.
  3. 3을 제외한 모든 3의 배수를 지운다.
  4. 4는 이미 지워졌으므로 skip한다.

  ...

소수 판별

Code Explanation

```
bool is_prime[MAX]; // 모두 true로 초기화
is_prime[1] = false;
for (int i = 2; i < MAX; i++){
    if(is_prime[i]){ // 만약 i가 소수라면
        for(int j = i * 2; j < MAX; j += i) is_prime[j] = false; // i를 제외한 i의 배수를 모두 지운다.
    }
}
```

소수 판별

- 에라토스테네스의 체의 시간복잡도는 아래와 같이 구할 수 있습니다.

$$\frac{n}{2}+\frac{n}{3}+\cdots+\frac{n}{n} \le nlogn$$

- 따라서 에라토스테네스의 체는 $O(nlogn)$안에 작동함을 알 수 있습니다.
- 실제 에라토스테네스의 체는 합성수는 건너뛰므로, $O(nloglogn)$안에 동작합니다.
- 이렇게 소수를 계산하면 소인수분해를 할 때 활용할 수 있습니다.
- DP[i] = (i의 가장 작은 약수) 로 정의한 후 DP를 채워두면 숫자 n의 소인수분해를 O(logn)안에 진행할 수 있습니다.

유클리드 알고리즘

- a와 b의 최대공약수는 a와 b의 공약수 중 가장 큰 공약수를 의미하고, gcd(a, b)로 표기합니다.
- 유클리드 알고리즘은 gcd(a, b)를 $O(\log(a+b))$안에 계산할 수 있는 매우 강력한 알고리즘입니다.
- 유클리드 알고리즘은 유클리드 호제법을 기초로 두고 있습니다.
- 유클리드 호제법 : $a \geq b$ 이고, a를 b로 나눈 나머지가 r일 때, $\gcd(a, b) = \gcd(b, r)$
- 예를 들어 a = 20, b = 15라면, gcd(20, 15) = gcd(15, 5)로 나타낼 수 있습니다.
- 유클리드 호제법을 b = 0이 될 때까지 계속 적용하여 a와 b의 최대공약수를 구하는 알고리즘이 유클리드 알고리즘 입니다.
- EX) gcd(20, 15) = gcd(15, 5) = gcd(5, 0) = 0

유클리드 알고리즘

Code Explanation

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

// 이 때 gcd(b, a % b)가 swap 역할을 하기 때문에 아래와 같이 swap을 생략하고 한 줄로 표현하기도 한다.
int gcd(int a, int b){
    return b == 0 ? a : gcd(b, a % b);
}
```

확장된 유클리드 알고리즘

- a와 b가 주어질 때, ax + by = gcd(a, b)가 되도록 하는 정수 x와 y를 찾는 알고리즘이 확장된 유클리드 알고리즘입니다. (x와 y는 항상 존재합니다. 단, 유일하지는 않습니다.)
- 예를 들어 a = 240, b = 46라 하면, gcd(a, b) = 2가 됩니다. 이 때 x와 y를 어떻게 구할 수 있을까요?
- gcd를 구하는 과정을 따라가봅시다.
- 240 = 46 ∗ 5 + 10

  46 = 10 ∗ 4 + 6

  10 = 6 ∗ 1 + 4

  6 = 4 ∗ 1 + 2

  4 = 2 ∗ 2

확장된 유클리드 알고리즘

- 오른쪽의 식을 이용해 x, y를 구할 수 있습니다.
- $10 = 240 * 1 + 46 * -5$
- $6 = 46 * 1 + 10 * -4$

  $= 46 * 1 + (240 * 1 + 46 * -5) * -4$

  $= 240 * -4 + 46 * 21$
- $4 = 10 * 1 + 6 * -1$

  $= 240 * 5 + 46 * (-26)$
- $2 = 6 * 1 + 4 * -1$

  $= 240 * (-9) + 46 * 47$
- 따라서 $x = -9$, $y = 47$일 때, $ax + by = gcd(a, b)$가 만족합니다.

$240 = 46 * 5 + 10$

$46 = 10 * 4 + 6$

$10 = 6 * 1 + 4$

$6 = 4 * 1 + 2$

$4 = 2 * 2$

확장된 유클리드 알고리즘

- 이 과정을 코드로 표현하기 위해 일반화를 진행해봅시다.
- 먼저 r1 = a, r2 = b, s1 = 1, s2 = 0, t1 = 0, t2 = 1로 초기화를 하고, 아래의 식이 만족하도록 r, s, t를 계속 update합니다.

$$r1 = s1 * a + t1 * b$$

$$r2 = s2 * a + t2 * b$$

- r1을 r2로 나눈 몫을 q라고 하면, 1번째 식에 2번째 식 * q 한 값을 빼주고, 두 식을 swap해주는 과정을 r2 = 0이 될 때까지 반복해주면 됩니다.

확장된 유클리드 알고리즘

Code Explanation

```
tuple <int, int, int> gcd(int a, int b){
    pair <int, int> r = {max(a, b), min(a, b)}, s = {1, 0}, t = {0, 1};
    while(b){
        int q = r.first / r.second;
        r = {r.second, r.first - q * r.second};
        s = {s.second, s.first - q * s.second};
        t = {t.second, t.first - q * t.second};
    }
    return make_tuple(r.first, s.first, t.first); // (gcd(a, b), x, y)를 반환한다.
}
```

확장된 유클리드 알고리즘

- 확장된 유클리드 알고리즘은 modular 연산에서 곱의 역원을 구할 때 자주 사용됩니다.
- (a*b) mod n = 1일 때 b는 a의 곱셈역이라고 합니다.
- 만약 gcd(a, n) = 1이면, a의 곱셈역은 존재함을 알 수 있습니다.
- 확장된 유클리드 알고리즘을 이용하여 $a * s + n * t = \gcd(a, n) = 1$을 만족하는 s, t를 구하고, 양 변에 mod n 을 취하면 $a * s \bmod n = 1$이 됩니다. 즉, s가 a의 곱셈역이 됩니다.
- 주로 n이 소수인 경우 ab mod n = 1이 성립하는 b를 구하기 위해 자주 사용됩니다.

연습 문제

- https://www.acmicpc.net/problem/1929 에라토스테네스의 체를 구현해보는 문제
- https://www.acmicpc.net/problem/1153 에라토스테네스의 체를 활용해보는 문제
- https://www.acmicpc.net/problem/2609 gcd와 lcm을 구해보는 문제
- https://www.acmicpc.net/problem/14565 확장된 유클리드 알고리즘을 구현해보는 문제