3. 기초 정수론

23 안해성

목차

- 나머지 연산
- 거듭 재곱
- 조합
- 소수 판별
- 유클리드 호제법

\+ 실수 연산에 도움 되는 함수

나머지 연산

modulo arithmetic

나머지란?

- 나눗셈 정리 :

$$a = bq + r \ (0 \ \leq r < |b|)$$

- 설명 :

  q는 b로 나눈 몫 q = a / b;

  r은 b로 나눈 나머지 r = a % b;

합동

- 정수 a, b, n이 있을 때, $n|(a-b)$ 이면,
  a와 b가 mod n에서 “합동”이다.

- $a \equiv b \ (mod\ n)$, 즉 a, b가 n으로 나눈 나머지가 같다는 뜻.

- e.g.)
  $5|(12-7)$, 5 / 5 = 1; $12 \equiv 7 \ (mod\ 5)$

나머지 연산 특징

- 정수 a, b, c 이 있을 때, 다음 식은 성립한다.
- $(a + b)\ mod\ c = (a\ mod\ c + b\ mod\ c)\ mod\ c$
- $(a - b)\ mod\ c = (a\ mod\ c - b\ mod\ c)\ mod\ c$
- $(a \times b)\ mod\ c = (a\ mod\ c \times b\ mod\ c)\ mod\ c$
- 나눗셈은 성립하지 않음.

17466. N! mod P (1)

- 문제 설명 :
  N과 소수 P가 주어지는데, N! mod P 구하는 문제.

- 가장 빠르게 생각해 볼만한 풀이 :
  반복문을 돌리며 N!을 구하고 그 값에 mod P를 하고 출력.
  하지만 N은 1억까지 들어오고 30!만 하더라도
  long long을 넘는다.

17466 모듈러를 사용한 풀이

- 나머지 연산의 특징 중,
  $(a \times b)\ mod\ c = (a\ mod\ c \times b\ mod\ c)\ mod\ c$
  을 이용 하면 된다.

- $N! = 1 \times 2 \times \cdots \times N$ 따라서
  $N! = \big((1 \times 2\ \%\ p) \times 3\ \%\ p\big) \ldots \times N\ \%\ p$

```cpp
#include <iostream>

using namespace std;
using LL = long long;

int main() {
    LL n, p;

    cin >> n >> p;

    LL ans = 1;
    for (int i = 2; i <= n; i++) {
        ans = (ans * i) % p;
    }

    cout << ans;

    return 0;
}
```

17466 코드

- 1e8 * 1e8하는 경우가 생기는데, 오버플로우 방지하기 위해 long long 사용.

거듭 재곱

exponentiation

거듭 제곱 일반적 구현

- O(n)으로 느리다고는 못 하지만, 수학의 공식에서 자주 쓰이는 제곱.

  더 빠르게는 구현 할 수 없을까?

- 지수는 음수를 제외한 정수에서만 생각한다.

```cpp
#include <iostream>

using namespace std;

int main() {
    int a, b;
    cin >> a >> b;

    int res = 1;
    for (int i = 0; i < b; i++) {
        res *= a;
    }

    cout << res;

    return 0;
}
```

거듭 제곱 O(log N)

- $a^b = \begin{cases} a^{b/2} \times a^{b/2}\ (2 \mid b) \\ a^{(b-1)/2} \times a^{(b-1)/2} \times a\ (2 \mid b) \\ 1\ (b = 0) \end{cases}$ 를 이용하면 된다.

- 구현은 재귀를 이용한 방법과, 그냥 반복문을 이용한 방법이 있다.

```cpp
#include <iostream>

using namespace std;

int myPow(int a, int b) {
    if (b == 0) return 1;

    int ret = myPow(a, b / 2);

    if (b % 2) {
        return ret * ret * a;
    }
    else {
        return ret * ret;
    }
}

int main() {
    int a, b;
    cin >> a >> b;

    cout << myPow(a, b);

    return 0;
}
```

거듭 제곱 코드

1620. 곱셈

- 문제 요약:
  정수 a, b, c가 주어진다. a^b mod c 를 구하라.

- 가장 빠르게 생각해 볼만한 풀이 :
  반복문 돌면서 a^b 구하고 mod c하기.

  하지만 b는 21억까지 나오고, O(N)이기 때문에 TLE.
  또 a^b는 long long 범위도 넘을 가능성이 다분하다.

1620 O(log n) 풀이

- 나머지 연산 특징 + 거듭 제곱 O(log n) 을 합쳐서 풀면 된다.

- 거듭 제곱도 N! 했을 때와 곱하는 수만 다를 뿐 같은 상황이므로,
  $(a \times b)\ mod\ c = (a\ mod\ c \times b\ mod\ c)\ mod\ c$
  사용하면 된다.

```cpp
#include <iostream>

using namespace std;
using LL = long long;

LL myPow(int a, int b, int c) {
    if (b == 0) return 1;

    LL ret = myPow(a, b / 2, c);

    if (b % 2) {
        return ((ret * ret)%c * a)%c;
    }
    else {
        return (ret * ret)%c;
    }
}

int main() {
    int a, b, c;
    cin >> a >> b >> c;

    cout << myPow(a, b, c);

    return 0;
}
```

1620 코드

- 2e9 * 2e9 정도의 연산이 될 수 있으므로 long long을 사용하였다.

이항 계수

binomial coefficient

이항 계수

- $\binom{n}{r}$으로 nCr, 즉 n개에서 r개 뽑는 경우의 수를 의미한다.

- 공식:
  $\binom{n}{r} = \frac{n!}{r!(n-r)!} = \frac{n\times(n-1)\times(n-2)\times\cdots\times(n-r+1)}{1\times2\times3\times\cdots\times r}$

  즉 $\frac{n}{1}\times\frac{n-1}{2}\times\frac{n-2}{3}\times\cdots\times\frac{n-r+1}{r}$ 로 구해도 된다.

+ 각 나누기의 결과는 정수이다.

11051. 이항 계수2

- 문제 요약 :
  정수 n, r이 주어진다. nCr mod 10,007 구하기.

- 가장 빠르게 생각해 볼만한 풀이 :
  전 문제들 처럼 nCr 공식 + 나머지 연산 특징?

  나머지 연산에는 나누기에 대한 특징은 없음.
  그렇다고 nCr먼저 구하기에는 연산 중간에 long long 범위 초과.

이항 계수 다른 공식

- $\binom{n}{r} = \binom{n-1}{r} + \binom{n-1}{r-1}$ 을 이용하면, 덧셈의 나머지 연산 특징 사용 가능.

- $\binom{n}{r} = \binom{n-1}{r} + \binom{n-1}{r-1}$ 을 말로 풀어 설명하면,
  n개를 선택하는데,
  (n번째를 선택하지 않고 r개 고르는 경우) +
  (n번째를 선택하고 r-1개를 고르는 경우)

11051 코드

```cpp
#include <iostream>

using namespace std;

int dp[1100][1100], prime = 10'007;

int comb(int n, int k) {
	if (dp[n][k]) return dp[n][k];
	if (n == k || k == 0) return 1;
	if (k == 1) return n;

	return dp[n][k] = (comb(n - 1, k) + comb(n - 1, k - 1)) % prime;
}

int main() {
	int n, k;
	cin >> n >> k;

	cout << comb(n, k);

	return 0;
}
```

- dp[n][k]에는 $\binom{n}{k}\ mod\ 10{,}007$ 값이 들어 있음.

소수 판별

prime

소수

- 소수는 자기자신과 1을 제외한 어떤 수로도
  나누어지지 않는 2이상의 자연수.

- 소수를 판별하는 방법 :
  자연수 n이 있을 때, [2, n-1]까지 수들로 나누어 떨어지는
  수가 있는지 확인하면 된다.

  있으면 소수X, 없으면 소수

O(n)보다 빠르게

- 소수인지 확인할 때, [2, n-1]까지 전부 돌아야 할까?

- 소수인지 확인할 때는 $\sqrt{n}$이하 까지만 확인해도 괜찮다.
  즉 O($\sqrt{n}$)에 판별 가능.

- 만약 n=pq에서 p가 $\sqrt{n}$ 이상의 수라면,
  "무조건" q는 $\sqrt{n}$ 이하의 수이다.

O($\sqrt{n}$) 소수 판별 코드

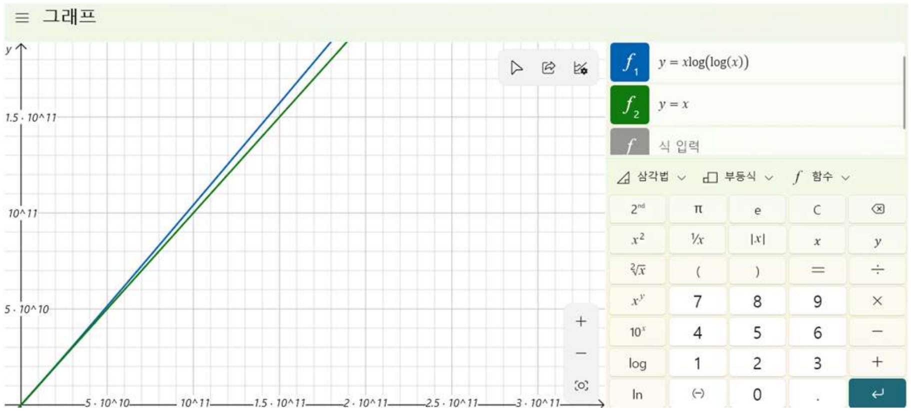
```cpp
#include <iostream>

using namespace std;

bool is_prime(int num) {
	for (int i = 2; i * i <= num; i++) {
		if (num % i == 0) return false;
	}

	return true;
}

int main() {
	int n;
	cin >> n;

	cout << is_prime(n);

	return 0;
}
```

- sqrt()함수는 느리니깐 i*i <= num으로 쓰는 게 좋다.
- 소수면 true, 아니면 false를 반환한다.
- 짝수, 3의 배수 제외하면 소수는 6k+1, 6k-1꼴인 것을 이용하면 O($\sqrt{n}$/3) 까지 줄일 수 있다.

에라토스테네스의 체

- 자연수 n이하의 소수를 전부 찾는 방법.
- n이하의 소수 p가 있을 때, p의 배수 중 n이하의 수를 전부 제거해가는 방식.

- 시간 복잡도는 $O(\sum_{x=2}^{n} \lfloor n/x \rfloor) \approx O(n \log \log n)$ 이고, 대충 O(n)에 가깝다고 한다.



```
bool chk[1'000'000];     //prime = false
vector<int> primes;

void sieve(int n) {
    chk[1] = true;

    for (int i = 2; i < n; i++) {
        if (chk[i]) continue;

        primes.push_back(i);
        for (int k = i + i; k < n; k += i) {
            chk[k] = true;
        }
    }
}


int main() {
    int n;
    cin >> n;

    sieve(n);

    for (const auto& p : primes) {
        cout << p << '₩n';
    }

    return 0;
}
```

에라토스테네스의 체 코드

- 정수 최대 1,000,000까지의 소수를 구하는 코드.

  소수는 primes 벡터에 저장된다.

소인수분해

- 자연수 n을 소수들의 곱으로 표현한 것.
- 어떤 자연수 n의 소인수분해는 유일하다.

- n 소인수분해 방법 :
  [2, n]의 수들을 차례대로 하나씩 보며, 나눠지지 않을 때까지 n을 나누기.
- e.g.) 84 -> 42 -> 21 -> 7 -> 1

- 시간 복잡도 O(n + log n)

O($\sqrt{n} + \log n$)으로 만들기

- 소수 판별 때와 마찬가지로 $\sqrt{n}$까지만 살펴보면 된다.
  $\sqrt{n}$ 까지 전부 보며 n을 나누었지만 n이 1이 아니라면, 그 n은 소인수가 된다.

$\sqrt{n}$ 초과의 소인수는 단 1개 밖에 존재 할 수 없기 때문이다.
a, b > $\sqrt{n}$; ab > n;

소인수분해 코드

```cpp
#include <iostream>
#include <vector>

using namespace std;

void prime_factor(int n) {
    for (int i = 2; i * i <= n; i++) {
        while (n % i == 0) {
            cout << i << ' ';
            n /= i;
        }
    }

    if (n != 1) {
        cout << n;
    }

    return;
}

int main() {
    int n;
    cin >> n;

    prime_factor(n);

    return 0;
}
```

유클리드 호제법

Euclidean algorithm

최대 공약수, 최소 공배수

- a = b = 0 이 아닌 정수 a, b가 있다.
- g|a, g|b를 만족하는 가장 큰 g : 최대 공약수
- a|L, b|L를 만족하는 가장 작은 L : 최소 공배수

- 최소 공배수와, 최대 공약수 관계 :
  L = a*b/g

  즉 최대 공약수만 알면 최소 공배수도 구할 수 있다.

최대 공약수(GCD)

- 두 정수 a, b의 최대 공약수를 GCD(a, b)라고 하자.
- GCD(a, b) = $\begin{cases} GCD(b, a\%b)\ (a \geq 1, b\ \geq 1) \\ GCD(a, 0) = a\ (a \geq 1, b = 0) \end{cases}$

- 모든 수는 0을 나눌 수 있다.

- 시간 복잡도는 O(log ab)

GCD 코드

```cpp
#include <iostream>

using namespace std;

int gcd(int a, int b) {
    int tmp;
    while (a % b) {
        tmp = a % b;
        a = b;
        b = tmp;
    }

    return b;
}

int main() {
    int n, m;
    cin >> n >> m;

    cout << gcd(n, m);

```

과제

- 1978.소수찾기 (소수판별)
- 5347.LCM (gcd)
- 16563.어려운 소인수분해
  (에라토스테네스 체+ 소인수분해 응용)