

#1. 사칙 연산

나정휘

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

목차

나눗셈과 합동식

배수 판정법

거듭 제곱의 빠른 계산

이항 계수

C/C++의 나눗셈 연산

나눗셈과 합동식

나눗셈 정리 (Division Algorithm)

- 정수 a 와 0이 아닌 정수 b 가 있을 때, $a = bq + r, 0 \leq r < |b|$ 를 만족하는 정수 q, r 은 유일함
 - 이때 q 를 몫, r 을 나머지라고 부름
 - 8을 3으로 나누었을 때의 몫은 2, 나머지는 2 ($8 = 3 \times 2 + 2$)
 - 8을 -3으로 나누었을 때의 몫은 -2, 나머지는 2 ($8 = (-3) \times (-2) + 2$)
 - -8을 3으로 나누었을 때의 몫은 -3, 나머지는 1 ($-8 = 3 \times (-3) + 1$)
 - -8을 -3으로 나누었을 때의 몫은 3, 나머지는 1 ($-8 = (-3) \times 3 + 1$)

나눗셈과 합동식

약수와 배수

- 정수 a, b 에 대해 $b = an$ 을 만족하는 정수 n 이 존재하면 a 는 b 의 약수, b 는 a 의 배수
 - $a|b$: a 가 b 를 나눈다, b 는 a 로 나누어진다.
 - 정의에 의해 모든 정수는 0을 나눌 수 있음
- $a \neq b \neq 0$ 인 정수 a, b 에 대해, $g|a, g|b$ 를 만족하는 가장 큰 자연수 g : 최대공약수
- $a|l, b|l$ 를 만족하는 가장 작은 자연수 l : 최소공배수
- a 와 b 의 최대공약수가 1이면 a 와 b 는 서로소
- $ab = gl$
 - $a = ga', b = gb'$ 라고 하면 a' 과 b' 은 서로소
 - $l = ga'b'$ 이므로 $ab = g^2a'b' = gl$ 임
 - $l = ab/g$

나눗셈과 합동식

합동

- 정수 a, b 와 0이 아닌 정수 n 이 있을 때, $n|(a - b)$ 이면 a 와 b 가 $(\text{mod } n)$ 에서 합동
 - $a \equiv b (\text{mod } n)$
 - a 와 b 를 n 으로 나눈 나머지가 동일하다는 뜻

합동의 성질

- 반사성, 대칭성, 추이성
 - $a \equiv a (\text{mod } n)$
 - $a \equiv b (\text{mod } n)$ 이면 $b \equiv a (\text{mod } n)$
 - $a \equiv b (\text{mod } n)$ 이고 $b \equiv c (\text{mod } n)$ 이면 $a \equiv c (\text{mod } n)$
- 사칙연산
 - $a \equiv b (\text{mod } n)$ 이면
 - 두 정수 $c \equiv d (\text{mod } n)$ 에 대해, $a \pm c \equiv b \pm c (\text{mod } n)$
 - 0이 아닌 두 정수 $c \equiv d (\text{mod } n)$ 에 대해, $ac \equiv bc (\text{mod } n)$
 - 나눗셈은 성립 안 함
 - $9 \equiv 12 (\text{mod } 3), 9/3 \not\equiv 12/3 (\text{mod } 3)$

배수 판정법

기본적인 지식

- 모든 정수 $a = \overline{a_n a_{n-1} a_{n-2} \cdots a_2 a_1 a_0}$ 은 $\sum a_i 10^i$ 로 나타낼 수 있음
- $a \equiv \overline{a_{k-1} a_{k-2} \cdots a_1 a_0} (mod 10^k)$
 - $i \geq k$ 인 항은 모두 0과 합동
- 따라서 $a \equiv a_0 (mod 10)$
- 마찬가지로 $10 \equiv 0 (mod 2)$ 이므로 $a \equiv a_0 (mod 2)$

- 10의 배수: 일의 자리 수가 0인 수
- 2의 배수: 일의 자리가 짝수(0, 2, 4, 6, 8)인 수

배수 판정법

4, 8, 16의 배수 판정법

- $100 \equiv 0 \pmod{4}$
- $\sum a_i 10^i$ 에서 $i \geq 2$ 인 항은 모두 100의 배수이므로 $a \equiv \overline{a_1 a_0} \pmod{4}$
- 마찬가지로 $1000 \equiv 0 \pmod{8}$ 이므로 $a \equiv \overline{a_2 a_1 a_0} \pmod{8}$
- $10000 \equiv 0 \pmod{16}$ 이므로 $a \equiv \overline{a_3 a_2 a_1 a_0} \pmod{16}$

- 4의 배수: 마지막 두 자리(100으로 나눈 나머지)가 4의 배수인 수
- 8의 배수: 마지막 세 자리가 8의 배수인 수
- 16의 배수: 마지막 네 자리가 16의 배수인 수

배수 판정법

3, 9의 배수 판정법

- 3의 배수: 모든 자리 수의 합이 3의 배수인 수
- 9의 배수: 모든 자리 수의 합이 9의 배수인 수
- $10 \equiv 1 \pmod{3}, 10 \equiv 1 \pmod{9}$
- 따라서 $a \equiv \sum a_i 10^i \equiv \sum a_i 1^i \equiv \sum a_i \pmod{3}, a \equiv \sum a_i \pmod{9}$

배수 판정법

11의 배수 판정법

- 11의 배수: 뒤에서부터 교대로 더하고 빼 수가 11의 배수인 수
 - ex. 30987은 $7-8+9-0+3 = 11$ 이므로 11의 배수
- $10 \equiv -1 \pmod{11}$
- 따라서 $a \equiv \overline{a_n a_{n-1} a_{n-2} \cdots a_2 a_1 a_0} \equiv \sum a_i 10^i \equiv \sum a_i (-1)^i \equiv a_0 - a_1 + a_2 - a_3 + \cdots \pmod{11}$

배수 판정법

7, 11, 13의 배수 판정법

- 7, 11, 13의 배수: 뒤에서부터 세 자리씩 교대로 더하고 뺀 수가 7, 11, 13의 배수인 수
 - ex. 85'002'918는 $918 - 002 + 085 = 1001$ 이고, $1001 = 7 * 11 * 13$ 이므로 7, 11, 13의 배수
- $1000 \equiv -1 \pmod{1001}$
- $a \equiv (-1)^0 \overline{a_2 a_1 a_0} + (-1)^1 \overline{a_5 a_4 a_3} + (-1)^2 \overline{a_8 a_7 a_6} + \dots \equiv \sum (-1)^i \overline{a_{3k+2} a_{3k+1} a_{3k}} \pmod{1001}$
- $7 \mid 1001$ 이므로 $a \equiv \sum (-1)^i \overline{a_{3k+2} a_{3k+1} a_{3k}} \pmod{7}$
- 마찬가지로 $11 \mid 1001, 13 \mid 1001$ 이므로 $\pmod{11}, \pmod{13}$ 에서도 성립

거듭제곱

목표: 음이 아닌 정수 a, b, c 에 대해 $a^b \pmod c$ 를 구하는 것

- $b = 0$ 이면 $a^b = 1$
- $b \geq 1$ 이면 두 가지 경우로 나뉘서 생각할 수 있음
 - $2 \mid b$ 이면 $a^b = a^{b/2} \times a^{b/2}$
 - $2 \nmid b$ 이면 $a^b = a \times a^{(b-1)/2} \times a^{(b-1)/2}$
- 곱셈 연산을 할 때마다 c 로 나눈 나머지를 취하면
- $a < c$ 일 때 계산 과정 도중에 나오는 값은 c^2 미만

거듭제곱

BOJ 1629. 곱셈

- $2147483647 (= 2^{31} - 1)$ 이하의 자연수 a, b, c 가 주어졌을 때
- $a^b \bmod c$ 를 구하는 문제
- 계산 도중에 int 범위를 넘어갈 수 있으므로 long long 사용
- Pow 함수의 호출 횟수: $\lfloor \log_2 b \rfloor + 2$ 번

```
● ● ●

#include <bits/stdc++.h>
using namespace std;
using ll = long long;

ll Pow(ll a, ll b, ll c){
    if(b == 0) return 1;
    ll half = Pow(a, b / 2, c);
    if(b % 2 == 0) return half * half % c;
    else return a * half % c * half % c;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    ll a, b, c;
    cin >> a >> b >> c;
    cout << Pow(a % c, b, c);
}
```

거듭제곱

BOJ 1629. 곱셈

- $b = 5$ 일 때 b 를 이진법으로 나타내 보면 $101_{(2)}(2^2 + 2^0)$
- a^5 는 $a^4 \times a^1$ 로 나타낼 수 있음
- $a^1, a^2, a^4, a^8, \dots$ 를 알고 있다면 최대 $\log_2 b$ 번의 곱셈으로 a^b 를 구할 수 있음



```
ll Pow(ll a, ll b, ll c){
    ll res = 1;
    while(b > 0){
        if(b % 2 == 1) res = res * a % c;
        b /= 2;
        a = a * a % c;
    }
    return res;
}
```

거듭제곱

BOJ 11819. The Shortest does not Mean the Simplest

- 똑같은 문제인데 $1 \leq a, b, c \leq 10^{18}$
 - 곱셈을 하는 순간 long long 범위를 넘어감
- 거듭제곱과 동일한 방법으로 $a \times b$ 를 $\log_2 b$ 번의 덧셈으로 구하면 됨
 - $a, b < c$ 일 때 $ab < c^2$ 이지만 $a + b < 2c$ 라서 long long 범위를 넘어가지 않음

```
ll Add(ll a, ll b, ll c){ return (a + b) % c; }

ll Mul(ll a, ll b, ll c){
    ll res = 0;
    while(b > 0){
        if(b % 2 == 1) res = Add(res, a, c);
        b /= 2; a = Add(a, a, c);
    }
    return res;
}

ll Pow(ll a, ll b, ll c){
    ll res = 1;
    while(b > 0){
        if(b % 2 == 1) res = Mul(res, a, c);
        b /= 2; a = Mul(a, a, c);
    }
    return res;
}
```

거듭제곱

BOJ 11819. The Shortest does not Mean the Simplest

- 똑같은 문제인데 $1 \leq a, b, c \leq 10^{18}$
- 채점 환경에 따라 `__int128_t`와 `__uint128_t`를 사용할 수 있음
- 하지만 `cin`, `cout`으로 입출력은 안 되므로 적절히 처리해야 함
 - 입출력 범위가 `long long` 이내라면 입출력만 `long long`으로 처리
 - 또는 문자열 $\leftrightarrow$ `__int128_t` 변환 구현

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
using L = __int128_t;

L Pow(L a, L b, L c){
    L res = 1;
    while(b > 0){
        if(b % 2 == 1) res = res * a % c;
        b /= 2; a = a * a % c;
    }
    return res;
}

int main(){
    ll a, b, c; cin >> a >> b >> c;
    cout << (ll)Pow(a, b, c);
}
```

거듭제곱

BOJ 10830. 행렬 제곱

- 행렬의 거듭제곱도 계산할 수 있음
- 행렬 곱셈의 항등원은 단위 행렬

```
vector<vector<int>> Mul(vector<vector<int>> a, vector<vector<int>> b){
    int n = a.size();
    vector<vector<int>> c(n, vector<int>(n));
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            for(int k=0; k<n; k++){
                c[i][j] = (c[i][j] + a[i][k] * b[k][j]) % 1000;
            }
        }
    }
    return c;
}

vector<vector<int>> Pow(vector<vector<int>> a, long long b){
    int n = a.size();
    vector<vector<int>> res(n, vector<int>(n));
    for(int i=0; i<n; i++) res[i][i] = 1;
    while(b > 0){
        if(b % 2 == 1) res = Mul(res, a);
        b /= 2; a = Mul(a, a);
    }
    return res;
}
```

이항 계수

BOJ 11050. 이항 계수 1

- $\binom{n}{r} = \frac{n!}{r!(n-r)!}$
 - n 개의 원소 중 r 개를 선택하는 경우의 수
 - $(x+1)^n$ 에서 x^r 의 계수
- 이항 계수를 구하는 가장 단순한 방법
 - $n!, r!, (n-r)!$ 을 각각 구한 다음 나눗셈
 - $n > 20$ 이면 64비트 정수형으로 $n!$ 을 저장할 수 없음



```
ll Factorial(int n){
    ll res = 1;
    for(int i=1; i<=n; i++) res *= i;
    return res;
}

ll Choose(int n, int r){
    return Factorial(n) / Factorial(r) / Factorial(n-r);
}
```

이항 계수

BOJ 11050. 이항 계수 1

- $\binom{n}{r} = \frac{n!}{r!(n-r)!} = \frac{n \times (n-1) \times (n-2) \times \dots \times (n-r+2) \times (n-r+1)}{1 \times 2 \times 3 \times \dots \times (r-1) \times r}$
 - n 곱하고 1 나누고
 - n-1 곱하고 2 나누고
 - ...
 - (n-r+1) 곱하고 r 나누고
- 계산 과정에서 나오는 수는 모두 정수
 - 연속한 정수 n개의 곱은 n!의 배수
 - $\frac{1}{n!}(k+1)(k+2)\dots(k+n) = \binom{k+n}{n}$ 이고, $\binom{k+n}{n}$ 은 정수이기 때문
- $r \leq \frac{n}{2}$ 이면 항상 $n \times \binom{n}{r}$ 보다 작거나 같은 값만 봐도 됨



```
ll Choose(int n, int r){
    ll res = 1;
    for(int i=0; i<r; i++) res = res * (n - i) / (i + 1);
    return res;
}
```

이항 계수

BOJ 11051. 이항 계수 2

- $\binom{n}{r} \bmod 10007$ 을 계산하는 문제
- 나눗셈이 들어가면 나머지 연산을 사용할 수 없음
- 나눗셈 없이 이항 계수를 계산하는 방법
 - $\binom{n}{r} = \binom{n-1}{r} + \binom{n-1}{r-1}$
 - (n번째 원소 선택하지 않고 1..n-1에서 r개 선택) + (n번째 원소 선택하고 1..n-1에서 r-1개 선택)
 - 덧셈만 이용해서 이항 계수를 구할 수 있음
- 파스칼의 삼각형

```
int C[1010][1010], MOD = 10'007;
void Calc(int n){
    C[0][0] = 1;
    for(int i=1; i<=n; i++){
        C[i][0] = 1;
        for(int j=1; j<=i; j++){
            C[i][j] = (C[i-1][j] + C[i-1][j-1]) % MOD;
        }
    }
}
```

이항 계수

BOJ 11051. 이항 계수 2

- $\binom{n}{r} \bmod 10007$ 을 계산하는 문제
- 나눗셈이 들어가면 나머지 연산을 사용할 수 없음
- 나눗셈 없이 이항 계수를 계산하는 방법
 - $\binom{n}{r} = \binom{n-1}{r} + \binom{n-1}{r-1}$
 - $0 \leq r_1, r_2 < M$ 이면 $0 \leq r_1 + r_2 < 2M$
 - 따라서 $r_1 + r_2 \geq M$ 일 때만 $r_1 + r_2 - M$ 사용하면 됨

```
int C[1010][1010], MOD = 10'007;
void Calc(int n){
    C[0][0] = 1;
    for(int i=1; i<=n; i++){
        C[i][0] = 1;
        for(int j=1; j<=i; j++){
            C[i][j] = C[i-1][j] + C[i-1][j-1];
            if(C[i][j] >= MOD) C[i][j] -= MOD;
        }
    }
}
```

C/C++의 나눗셈 연산

C/C++의 나눗셈 연산

- a, b가 정수일 때 $(a / b) * b + (a \% b) = a$ 를 만족하는 몫과 나머지를 반환함
- b가 a의 약수가 아니면 몫 0에 가까워지도록 자름 (truncation towards zero)

- $8 / 3 = 2$	$8 \% 3 = 2$	$2 * 3 + 2 = 8$
- $8 / (-3) = -2$	$8 \% (-3) = 2$	$(-2) * (-3) + 2 = 8$
- $(-8) / 3 = -2$	$(-8) \% 3 = -2$	$(-2) * 3 + (-2) = -8$
- $(-8) / (-3) = 2$	$(-8) \% (-3) = -2$	$2 * (-3) + (-2) = -8$

- 나머지를 $0 \leq r < |b|$ 범위에 넣고 싶을 때는
 - $r = r + (r \geq 0 ? 0 : |b|)$
 - 또는 $r = (r + |b|) \% |b|$

C/C++의 나눗셈 연산

BOJ 17466. N! mod P (1)

- N과 P가 주어지면 N!을 P로 나눈 나머지를 구하는 문제
- $N, P \leq 10^8$ 이므로 계산 과정 도중에 최대 10^{16} 까지의 수를 볼 수 있음
- long long을 사용하면서, 곱셈 연산을 할 때마다 나머지를 취하면 됨



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

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    ll N, P, R = 1;
    cin >> N >> P;
    for(int i=1; i<=N; i++) R = R * i % P;
    cout << R;
}
```

C/C++의 나눗셈 연산

C/C++의 나눗셈 연산

- floor와 ceil 직접 구현하기
 - math.h (또는 cmath)에 있는 floor/ceil은 부동 소수점 자료형을 사용하기 때문에 정확하지 않음
 - 정수 간의 나눗셈의 결과를 버림/올림하는 것은 정수만 사용해서 정확하게 수행할 수 있음
- p / q 에서 항상 $q > 0$ 이라고 가정하자.
 - $q < 0$ 이면 p 와 q 에 각각 -1 곱하면 됨
- $p \geq 0$ 인 경우
 - 버림: p / q
 - 올림: $(p + q - 1) / q$
- $p < 0$ 인 경우
 - 버림: $(p - q + 1) / q$
 - 올림: p / q



```
int floor(int p, int q){
    if(q < 0) p = -p, q = -q;
    return p >= 0 ? p / q : (p - q + 1) / q;
}

int ceil(int p, int q){
    if(q < 0) p = -p, q = -q;
    return p >= 0 ? (p + q - 1) / q : p / q;
}
```