N! mod P

배준재

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
3 초 (추가 시간 없음)	1024 MB	2827	1471	1079	54.827%

문제

양의 정수 N과, N보다 큰 소수 P가 주어질 때, N!을 P로 나눈 나머지를 구하여라.

입력

첫째 줄에 N과 P가 공백으로 구분되어 주어진다.

출력

N!을 P로 나눈 나머지를 구하여라.

제한

- $1 \le N < P \le 10^8$
- P는 소수

예제 입력 1 복사

```
4 7
```

예제 출력 1 복사

```
3
```

예제 입력 2 복사

```
99999988 99999989
```

예제 출력 2 복사

```
99999988
```

How to solve?

- N까지 배열을 돌면서 모듈러 연산을 하자!

```java
import java.util.*;
public class Main {
    public static void main(String[] args){
        Scanner scanner = new Scanner(System.in);
        long n = scanner.nextLong();
        long p = scanner.nextLong();
        long num=1;
        for(int i=2;i<n+1;i++){
            num=(num*i)%p;
        }
        System.out.println(num%p);
    }
}
```

91247301	bjj3141592	17466	맞았습니다!!	17668 KB	1440 ms	Java 11 / 수정	344 B	2분 전

너무 느리지 않나? 다시 한 번 조건을 보자

?

제한

- $1 \le N < P \le 10^8$
- P는 소수

윌슨의 정리

$$(P-1)! \equiv -1 \pmod{P}$$

증명

P가 소수이다.
우리는 집합 A ={1,2,3,…,P-1}을 생각해 볼 수 있다.

여기서 임의의 수 x를 생각해보자.
P는 소수이므로 모든 A의 원소와 서로소이다.

증명

이제 x의 역원 x'를 보자.
역원이라 함은 xx'≡1 (mod P)를 만족하는 x'이다.

1. 역원은 0일 수 없는 것이 자명하다.
2. 집합 A는 0이 아닌 P-1까지의 모든 수를 포함하므로
   x'는 반드시 집합A안에 있다.
3. 역원이 자기 자신인 경우 $x^2 \equiv 1$(mod P)이다.

증명

$x^2 \equiv 1$(mod P)에 따라 P|$x^2 - 1$이다.

그럼 P|(x-1)(x+1)임을 알 수 있고

P가 소수이므로 x-1,x+1과 서로소이다.

따라서 P|x-1 or P|x+1이다.

즉 $\mathrm{x} \equiv 1$(mod P), $\mathrm{x} \equiv -1$(mod P)이다.

집합A 내에서 이를 만족하는 수는 1, P-1이다.

증명

이제 집합 B={2,3,…,P-2}를 생각해보자 앞서 1,P-1이 빠졌으므로 집합 B안의 원소들은 서로를 역원으로 가진다.

$$(2 \cdot x_2) \times (3 \cdot x_3) \times \cdots \times ((p-2) \cdot x_{p-2}) \equiv 1 \pmod{p}$$

이제 전체 곱을 다시 보자

(P-1)! = 1x2x…x(P-1)에서 위 수식으로 인해

(P-1)! ≡ -1 (mod P)이다.

QED

적용

$N<P$ 이므로 N!은 (P-1)!의 약수이다.

(P-1)! = N! x $\prod_{N+1}^{P-1} i$ 이다.

이때 $\prod_{N+1}^{P-1} i$의 역원 $\prod_{N+1}^{P-1} i^{-1}$ 을 구하고 mod P하면

N! ≡ (P-1)! x $\prod_{N+1}^{P-1} i^{-1}$ mod P이다.

윌슨의 정리에 따라 N! ≡ -1x$\prod_{N+1}^{P-1} i^{-1}$ mod P가 된다.

적용

N! ≡ -1x $\prod_{N+1}^{P-1} i^{-1}$ mod P 에서

j= P-1이라 두면

이때 $\prod_{N+1}^{P-1} i \equiv \prod_{1}^{P-N-1}(P-j) \equiv (-1)^{P-N-1}(P-N-1)!\, mod\ P$

따라서 N! ≡ $(-1)^{P-N}(P-N-1)!^{-1}\ mod\ P$ 이다.

역원

페르마의 소정리

$x^{-1} \equiv x^{P-2}\ mod\ P$ 를 이용해 구할 수 있다.

구현

페르마의 소정리

```
static long modExp(long a, long b, long m) {
        long result = 1;
        a %= m;
        while (b > 0) {
            if ((b & 1) == 1) {
                result = (result * a) % m;
            }
            a = (a * a) % m;
            b >>= 1;
        }
        return result;
    }
```

구현

```
if (N <= P - N - 1) {
        long fact = 1;
        for (long i = 1; i <= N; i++) {
            fact = (fact * i) % P;
        }
        ans = fact;
    } else {
        long m = P - N - 1;
        long fact = 1;
        for (long i = 1; i <= m; i++) {
            fact = (fact * i) % P;
        }
        // (P-N-1)!의 모듈러 역원을 페르마의 소정리로 계산
        long invFact = modExp(fact, P - 2, P);
        // 부호 계산: (-1)^(P-N)
        long sign = ((P - N) % 2 == 1) ? (P - 1) : 1;
        ans = (sign * invFact) % P;
    }
```

결과

시간 복잡도

N!을 계산하는데 O(N) 또는 O(P - N - 1)

모듈러 연산 O(log P)이므로

O(min(N, P - N - 1)) + O(log P)이고 최악의 경우O(P)이다.

91247301	bjj3141592	17466	맞았습니다!!	17668 KB	1440 ms	Java 11 / 수정	344 B	58분 전

91208676	bjj3141592	17466	맞았습니다!!	18120 KB	796 ms	Java 11 / 수정	1775 B	18시간 전

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
3 초 (추가 시간 없음)	1024 MB	902	251	93	34.701%

문제

양의 정수 N과, N보다 큰 소수 P가 주어질 때, N!을 P로 나눈 나머지를 구하여라.

입력

첫째 줄에 N과 P가 공백으로 구분되어 주어진다.

출력

N!을 P로 나눈 나머지를 구하여라.

제한

- $1 \leq N < P \leq 10^9$
- P는 소수

윌슨의 정리로 풀 수 없다.

91217839	2 bjj3141592	5 17467	시간 초과				Python 3 / 수정	2559 B	16시간 전
91217575	2 bjj3141592	5 17467	틀렸습니다	(6%)			Python 3 / 수정	3047 B	16시간 전
91217535	2 bjj3141592	5 17467	시간 초과	(1%)			C++17 / 수정	1162 B	16시간 전
91217337	2 bjj3141592	5 17467	시간 초과	(6%)			PyPy3 / 수정	1429 B	16시간 전
91217150	2 bjj3141592	5 17467	시간 초과	(1%)			C++17 / 수정	2127 B	16시간 전
91216996	2 bjj3141592	5 17467	시간 초과				Python 3 / 수정	4039 B	16시간 전
91216767	2 bjj3141592	5 17467	시간 초과	(6%)			PyPy3 / 수정	1568 B	16시간 전
91215621	2 bjj3141592	5 17467	시간 초과	(6%)			Java 11 / 수정	2889 B	16시간 전
91214885	2 bjj3141592	5 17467	시간 초과	(6%)			Java 11 / 수정	1894 B	16시간 전
91214325	2 bjj3141592	5 17467	시간 초과	(6%)			C++17 / 수정	1315 B	17시간 전
91214100	2 bjj3141592	5 17467	시간 초과	(6%)			Java 11 / 수정	1743 B	17시간 전

다항식의 성질

다항식 $f_d = (dx+1)(dx+2)\ldots(dx+d)$를 생각해보자

1. f_d는 d차 다항식임이 자명하다.
2. $f_N(0) = N!\,, x = 0$을 넣으면 1x2x…xN이므로
3. $f_a(0)f_a(1)\ \ldots f_a(b-1)$=$f_{ab}(0)$이다.

$f_a(0)$ = 1x2x…x(a)

$f_a(1)$ = (a+1)x(a+2)x…x(a+a = 2a)

$f_a(2)$ = (2a+1)x(2a+2)x…x(2a+a =3a)

$f_a(b-1)$ = (a(b-1)+1)x(a(b-1)+2)x…x(a(b-1)+a = ab)이므로

4. $f_d(2x)f_d(2x+1) = f_{2d}(x)$이다. < 3번식을 변형

다항식의 성질

이 성질을 이용하면

N!은 $f_N(0)$을 구하면 된다.

적당한 v≈ $\sqrt{N}$ 을 잡아 $f_v(0), f_v(1), \dots, f_v(v)$를 구할 수 있다면

$f_{v(v+1)}(0)$을 계산할 수 있다.

나머지 v(v+1) +1부터 N까지는 직접 곱으로 구할 수 있다.

이 과정에서 T($\sqrt{N}$)이 걸린다.

Multipoint evaluation(다항식 다중계산)

앞서 구한 다항식이 같은 형태이기 때문에 multipoint evaluation을 사용할 수 있다.

다항식 다중계산이란 N차 다항식 f에 대해 Q개의 원소 x1,x2..xq를 O(max(N,Q)log(max(N,Q)))시간내 풀 수 있는 알고리즘이다.

다항식 곱셈

FFT를 사용한다.

$x_0, x_1, \dots, x_{N-1}$에 대해 $X_{0,}X_{1,\dots,}X_{N-1}$로 바꾸는 변환이다.

$X_k = \sum_{n=0}^{N-1} x_n w^{nk}$, k=0,…,N-1, $w^N = 1$

이것을 좀 더 풀어쓰면

$f(a) = x_0 a^0 + x_1 a^1 + \dots + x_{N-1} a^{N-1}$에서

a에 $w^0, w^1 \dots w^{N-1}$을 대입한 것이다.

Lagrange Interpolation

앞서 다항식 f가 있을 때 각 항을 쓸 수 있지만
$f(w^0), f(w^1), \ldots, f(w^{N-1})$이렇게 값을 들고 다녀도 된다.

두 N-1차 다항식 f와 M-1차 다항식 g가 있을 때
둘을 곱함으로서 $\mathrm{h(x)} \equiv \mathrm{f(x)g(x)}$는 N+M-1차 다항식이다.
여기서 우리는 N+M개 이상의 함수값을 알면 h를 유일하게 결정할 수 있다.

Lagrange Interpolation

이제, 우리는 임의의 다항식이 가지는 간단한 성질을 알아보려고 한다. N 을 편의상 짝수라고 가정하자.

$f(a) = x_0a^0 + x_1a^1 + \cdots + x_{N-1}a^{N-1}$ 일 때, 이를 홀수와 짝수로 나눈다면,

$E(a) = x_0a^0 + x_2a^1 + \cdots + x_{N-2}a^{N/2-1}$

$O(a) = x_1a^0 + x_3a^1 + \cdots + x_{N-1}a^{N/2-1}$

따라서 $\mathrm{f(a)} = \mathrm{E}(a^2) + \mathrm{aO}(a^2)$으로 표현된다.
이 과정으로 N-1차 다항식을
N/2 -1차 다항식의 표현으로 변환할 수 있다.

다중 계산

다중계산을 할 때, N−1차 다항식을 N개의 점에 대해서 구할 것이다. (N=2^k)

우리는 강력한 정리 하나를 이용할 것이다

$f(x)$를$(x-a)g(x)$로 나눈 나머지를 $r(x)$라 할 때,

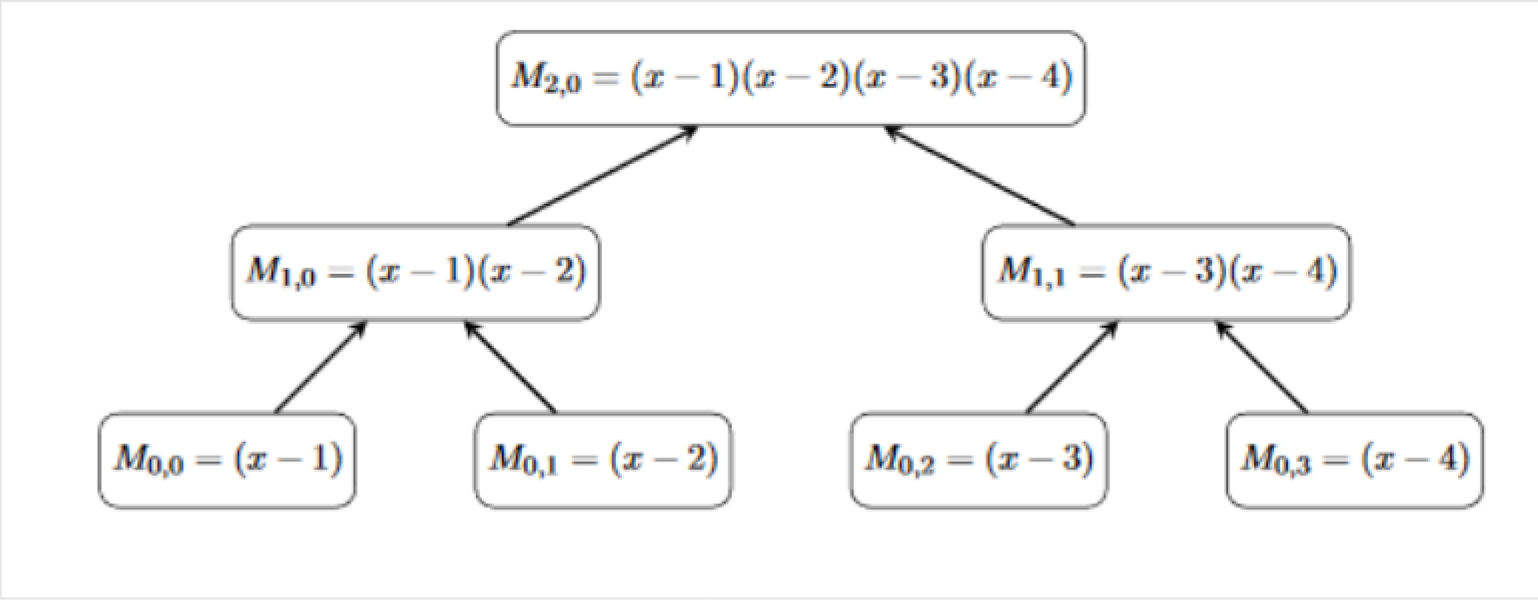
$f(a)$는 $\mathrm{r}(a)$와 같다. a_0

$f(x)$를$(x-a_0)(x-a_1)\left(x-a_{N/2-1}\right)$로 나눈 나머지를 구하고,

a_0, $a_1, \dots, a_{N/2-1}$을 대입한 값을 구하고

뒤도 마찬가지로 구할 수 있다.

다중계산



Polynomial tree

복소수 구현

```java
// 복소수 클래스 (실수부 re, 허수부 im)
static class Complex {  40개 사용 위치
    double re, im;  15개 사용 위치
    Complex(double re, double im) {  14개 사용 위치
        this.re = re;
        this.im = im;
    }
    Complex add(Complex o) { return new Complex( re: this.re + o.re,  im: this.im + o.im); }
    Complex subtract(Complex o) {  3개 사용 위치
        return new Complex( re: this.re - o.re,  im: this.im - o.im);
    }
    Complex multiply(Complex o) {  7개 사용 위치
        return new Complex( re: this.re * o.re - this.im * o.im,  im: this.re * o.im + this.im * o.re);
    }
    Complex scale(double factor) {  3개 사용 위치
        return new Complex( re: this.re * factor,  im: this.im * factor);
    }
    Complex conjugate() {  4개 사용 위치
        return new Complex(this.re, -this.im);
    }
}
```

모듈러 덧셈

```java
// 모듈러 덧셈
static long modAdd(long a, long b, long mod) {  4개 사용 위치
    a %= mod; b %= mod;
    long s = a + b;
    s %= mod;
    if(s < 0) s += mod;
    return s;
}
```

모듈러 곱셈

```java
// 모듈러 곱셈 (안전하게 계산; 두 수가 매우 클 때 오버플로우 방지)
static long modMul(long a, long b, long mod) {  17개 사용 위치
    a %= mod; b %= mod;
    long result = 0;
    while(b > 0) {
        if ((b & 1) == 1) {
            result = modAdd(result, a, mod);
        }
        a = modAdd(a, a, mod);
        b >>= 1;
    }
    return result % mod;
}
```

모듈러 거듭제곱

```java
// 모듈러 거듭제곱 (b의 거듭제곱을 mod에서 계산)
static long ipow(long a, long b, long mod) {  1개 사용 위치
    long res = 1 % mod;
    a %= mod;
    while (b > 0) {
        if ((b & 1) == 1)
            res = modMul(res, a, mod);
        a = modMul(a, a, mod);
        b >>= 1;
    }
    return res;
}
```

모듈러 역원

```java
// 모듈러 역원 (P가 소수이므로 Fermat의 소정리 이용)
static long modInv(long a, long mod) {  1개 사용 위치
    return ipow(a,  b: mod - 2, mod);
}
```

FFT 구현 (in-place Cooley-Turkey)

비트를 001-> 100으로 변환

```
static void fft(Complex[] a, boolean inv) {  4개 사용 위치
    int n = a.length;
    int j = 0;
    //비트 반전 001 -> 100
    for (int i = 1; i < n; i++) {
        int bit = n >> 1;
        while (j >= bit) {
            j -= bit;
            bit >>= 1;
        }
        j += bit;
        if (i < j) {
            Complex tmp = a[i];
            a[i] = a[j];
            a[j] = tmp;
        }
    }
```

FFT 구현 (in-place Cooley-Turkey)

짝수, 홀수를 저장

```
//파이값 구하는 법
double ang = 2 * Math.acos(-1) / n * (inv ? -1 : 1);
//대칭성을 이용해 절반만 저장
Complex[] roots = new Complex[n / 2];
for (int i = 0; i < n / 2; i++) {
    roots[i] = new Complex(Math.cos(ang * i), Math.sin(ang * i));
}
//2,4,8,16으로 커짐
for (int len = 2; len <= n; len <<= 1) {
    int step = n / len;
    for (int i = 0; i < n; i += len) {
        for (int k = 0; k < len / 2; k++) {
            Complex u = a[i + k];
            Complex v = a[i + k + len / 2].multiply(roots[step * k]);
            a[i + k] = u.add(v); // 짝수
            a[i + k + len / 2] = u.subtract(v); // 홀수
        }
    }
}
```

FFT 구현 (in-place Cooley-Turkey)

역변환

```
//역변환
if (inv) {
    for (int i = 0; i < n; i++) {
        a[i] = a[i].scale( factor: 1.0 / n);
    }
}
```

다항식 곱셈
실수부

```java
// FFT를 이용한 다항식 곱셈 (v, w의 계수를 mod에서 계산)
static long[] multiply(long[] v, long[] w, long mod) {  1개 사용 위치
    int n = 2;
    while (n < v.length + w.length) n <<= 1; // 가장가까운 2^n으로 맞춤
    Complex[] v1 = new Complex[n];
    Complex[] v2 = new Complex[n];
    Complex[] r1 = new Complex[n];
    Complex[] r2 = new Complex[n];
    for (int i = 0; i < n; i++) {
        if (i < v.length)
            v1[i] = new Complex( re: v[i] >> 15,  im: v[i] & 32767); // 상위 15비트 추출, 하위 15비트 추출32767은 15비트의 최대값
        else
            v1[i] = new Complex( re: 0,  im: 0);
        if (i < w.length)
            v2[i] = new Complex( re: w[i] >> 15,  im: w[i] & 32767);
        else
            v2[i] = new Complex( re: 0,  im: 0);
    }
    fft(v1,  inv: false);
    fft(v2,  inv: false);
```

다항식 곱셈

허수부

```java
//복소수 부분 연산
for (int i = 0; i < n; i++) {
    int j = (i == 0 ? 0 : n - i); //j는 반대쪽 인덱스
    // ans1 = (v1[i] + conj(v1[j])) * (0.5, 0)
    Complex ans1 = v1[i].add(v1[j].conjugate()).scale( factor: 0.5);
    // ans2 = (v1[i] - conj(v1[j])) * (0, -0.5)
    Complex ans2 = v1[i].subtract(v1[j].conjugate());
    ans2 = new Complex( re: 0.5 * ans2.im,  im: -0.5 * ans2.re);
    // ans3, ans4 for v2
    Complex ans3 = v2[i].add(v2[j].conjugate()).scale( factor: 0.5);
    Complex ans4 = v2[i].subtract(v2[j].conjugate());
    ans4 = new Complex( re: 0.5 * ans4.im,  im: -0.5 * ans4.re);

    r1[i] = ans1.multiply(ans3).add(ans1.multiply(ans4).multiply(new Complex( re: 0,  im: 1)));
    r2[i] = ans2.multiply(ans3).add(ans2.multiply(ans4).multiply(new Complex( re: 0,  im: 1)));
}
fft(r1,  inv: true); // 역 연산
fft(r2,  inv: true); // 역 연산
```

다항식 곱셈

실수부와 허수부합

```java
long[] ret = new long[n];
for (int i = 0; i < n; i++) {
    long av = Math.round(r1[i].re) % mod;//상위 30부분
    long bv = (Math.round(r1[i].im) + Math.round(r2[i].re)) % mod;//중15비트
    long cv = Math.round(r2[i].im) % mod;//하위15비트
    // (av << 30) + (bv << 15) + cv, mod mod
    long tmp = modMul(av, 1L << 30, mod);
    tmp = modAdd(tmp, modMul(bv, 1L << 15, mod), mod);
    tmp = modAdd(tmp, cv, mod);
    ret[i] = ((tmp % mod) + mod) % mod;
}
return ret;
```

라그랑주 보간법

```java
static long[] lagrange(long[] h, long P) {
    int d = h.length - 1;
    if (d < 0) return new long[0];
    int size = 4 * d + 2;
    long[] fact = new long[size]; // i! mod p
    long[] invfact = new long[size]; //i! ^ -1 mod p
    fact[0] = 1;
    for (int i = 1; i < size; i++) {
        fact[i] = modMul(i, fact[i - 1], P);
    }
    invfact[size - 1] = modInv(fact[size - 1], P);
    for (int i = size - 2; i >= 0; i--) {
        invfact[i] = modMul(invfact[i + 1], i + 1, P);
    }
    long[] f = new long[d + 1];
    for (int i = 0; i <= d; i++) {
        f[i] = h[i] % P;
        f[i] = modMul(invfact[i], f[i], P);
        f[i] = modMul(invfact[d - i], f[i], P);
        if ((d - i) % 2 == 1)
            f[i] = (P - f[i]) % P;
        if (f[i] == P) f[i] = 0;
    }
```

라그랑주 보간법

```java
long[] inv = new long[size];
for (int i = 1; i < size; i++) {
    inv[i] = modMul(fact[i - 1], invfact[i], P);
}
long[] g = new long[size];
g[d + 1] = 1;
for (int j = 0; j <= d; j++) {
    g[d + 1] = modMul(g[d + 1], (d + 1 - j), P);
}
for (int i = d + 2; i < size; i++) {
    g[i] = modMul(g[i - 1], modMul(i, inv[i - d - 1], P), P);
}
long[] conv = multiply(f, inv, P);
long[] ret = new long[size];
for (int i = 0; i <= d; i++) {
    ret[i] = h[i] % P;
}
for (int i = d + 1; i < size; i++) {
    ret[i] = modMul(g[i], conv[i], P);
}
return ret;
```

라그랑주 보간법

```java
// squarepoly: lagrange 보간 후 짝수, 홀수 인덱스 계수를 곱함
static long[] squarepoly(long[] poly, long P) {  1개 사용 위치
    long[] ss = lagrange(poly, P);
    int newSize = ss.length / 2;
    long[] ret = new long[newSize];
    for (int i = 0; i < newSize; i++) {
        ret[i] = modMul(ss[2 * i], ss[2 * i + 1], P);
    }
    return ret;
}
```

메인

```java
// 팩토리얼을 미리 구함
long d = 1;
long[] factPart = new long[] { 1 % P, 2 % P };
while (N > d * (d + 1)) {
    factPart = squarepoly(factPart, P);
    d *= 2;
}

long bucket = N / d;
long ans = 1;
//fact v까지 연산
for (int i = 0; i < bucket; i++) {
    ans = modMul(ans, factPart[i], P);
}
// v 이후 곱
for (long i = bucket * d + 1; i <= N; i++) {
    ans = modMul(ans, i, P);
}
bw.write(ans + "\n");
```

흐름

홀수부와 짝수부로 나눠 라그랑주 보간법을 사용한다.
이때 다중함수로 연산하는데
다중함수에서 FFT를 사용한다.

결과

다중계산에서 $\sqrt{N}logN$ 시간이 걸린다.

91234951	2 bjj3141592	5 17467	맞았습니다!!	320008 KB	2208 ms	Java 11 / 수정	9042 B	12시간 전