

# 杜教筛

## 积性函数

在数论题目中，常常需要根据一些积性函数的性质，求出一些式子的值。

**积性函数**：对于函数  $f(n)$  有  $f(1)=1$  且对于所有互质的  $a$  和  $b$  总有  $f(ab)=f(a)f(b)$  则称  $f(n)$  为积性函数。

常见的积性函数有：

- $\sigma_0(n)=d(n)=\sum_{d|n} 1$
- $\sigma(n)=\sum_{d|n} d$
- $\varphi(n)=\sum_{i=1}^n [\gcd(x,i)=1]$
- $\mu(n)=\begin{cases} 1 & n=1 \\ (-1)^k & \prod_{i=1}^k q_i=1 \text{ 且 } \max\{q_i\}>1 \end{cases}$

积性函数有如下性质：

若  $f(n)$  和  $g(n)$  为积性函数，则：

- $h(n)=f(n^p)$
- $h(n)=f^p(n)$
- $h(n)=f(n)g(n)$
- $h(n)=\sum_{d|n} f(d)g(\frac{n}{d})$

中的  $h(n)$  也为积性函数。

在莫比乌斯反演的题目中，往往要求出一些数论函数的前缀和，利用杜教筛可以快速求出这些前缀和。

## 杜教筛

杜教筛被用来处理数论函数的前缀和问题。对于求解一个前缀和，杜教筛可以在低于线性时间的复杂度内求解

对于数论函数  $f$  要求我们计算  $S(n)=\sum_{i=1}^n f(i)$ 。

我们想办法构造一个  $S(n)$  关于  $S(\lfloor \frac{n}{i} \rfloor)$  的递推式

对于任意一个数论函数  $g$  必满足  $\sum_{i=1}^n \sum_{d|mid i} f(d)g(\frac{i}{d}) = \sum_{i=1}^n g(i)S(\lfloor \frac{n}{i} \rfloor)$   
 $\Leftrightarrow \sum_{i=1}^n (f * g)(i) = \sum_{i=1}^n g(i)S(\lfloor \frac{n}{i} \rfloor)$

略证：

$f(d)g(\frac{i}{d})$  就是对所有  $i \leq n$  的贡献，因此变换枚举顺序，枚举  $d, \frac{i}{d}$  分别对应新的  $i, j$   
 $\sum_{i=1}^n \sum_{d|mid i} f(d)g(\frac{i}{d}) = \sum_{i=1}^n \sum_{j=1}^{\lfloor \frac{n}{i} \rfloor} g(i)f(j)$   
 $= \sum_{i=1}^n g(i) \sum_{j=1}^{\lfloor \frac{n}{i} \rfloor} f(j)$

$= \sum_{i=1}^n S(\lfloor \frac{n}{i} \rfloor)$  那么可以得到递推式  $g(1)S(n) = \sum_{i=1}^n (f * g)(i) - \sum_{i=2}^n S(\lfloor \frac{n}{i} \rfloor)$  那么假如我们可以快速对  $\sum_{i=1}^n (f * g)(i)$  求和，并用数论分块求解  $\sum_{i=2}^n S(\lfloor \frac{n}{i} \rfloor)$  就可以在较短时间内求得  $g(1)S(n)$

## 问题一

P4213 模板】杜教筛 Sum

题目大意：求  $S_1(n) = \sum_{i=1}^n \mu(i)$  和  $S_2(n) = \sum_{i=1}^n \varphi(i)$  的值  $n \leq 2^{31} - 1$

### 莫比乌斯函数前缀和

由狄利克雷卷积，我们知道：

$$\mu * 1 = \epsilon$$

$$\epsilon(n) = \begin{cases} 1 & n=1 \\ 0 & \text{otherwise} \end{cases}$$

$$S_1(n) = \sum_{i=1}^n \mu(i) - \sum_{i=2}^n S_1(\lfloor \frac{n}{i} \rfloor)$$

$$= 1 - \sum_{i=2}^n S_1(\lfloor \frac{n}{i} \rfloor)$$

观察到  $\lfloor \frac{n}{i} \rfloor$  最多只有  $O(\sqrt{n})$  种取值，我们就可以应用整除分块（或称数论分块）来计算每一项的值了。

直接计算的时间复杂度为  $O(n^{\frac{3}{4}})$  考虑先线性筛预处理出前  $n^{\frac{2}{3}}$  项，剩余部分的时间复杂度为  $O(\int_0^{\frac{1}{3}} \sqrt{\frac{n}{x}} dx) = O(n^{\frac{2}{3}})$  对于较大的值，需要用 map 存下其对应的值，方便以后使用时直接使用之前计算的结果。

### 欧拉函数前缀和

当然也可以用杜教筛求出  $\varphi(x)$  的前缀和，但是更好的方法是应用莫比乌斯反演：

$$\sum_{i=1}^n \sum_{j=1}^n [\gcd(i,j)=1] = \sum_{i=1}^n \sum_{j=1}^n \sum_{d \mid i, d \mid j} \mu(d) = \sum_{d=1}^n \mu(d) \lfloor \frac{n}{d} \rfloor^2$$

观察到，只需求出莫比乌斯函数的前缀和，就可以快速计算出欧拉函数的前缀和了。时间复杂度  $O(n^{\frac{2}{3}})$

### 使用杜教筛求解

$$S(i) = \sum_{i=1}^n \varphi(i)$$

$$\varphi * 1 = \text{id} \quad \sum_{i=1}^n (\varphi \ast 1)(i) = \sum_{i=1}^n i \cdot S(\lfloor \frac{n}{i} \rfloor) \quad \sum_{i=1}^n iD(i) = \sum_{i=1}^n i \cdot S(\lfloor \frac{n}{i} \rfloor)$$

$$\frac{1}{2}n(n+1) = \sum_{i=1}^n \left\lfloor \frac{n}{i} \right\rfloor$$

$$S(n) = \frac{1}{2}n(n+1) - \sum_{i=2}^n \left\lfloor \frac{n}{i} \right\rfloor$$

参考代码：

```

#include <algorithm>
#include <cstdio>
#include <cstring>
#include <map>
using namespace std;
const int maxn = 2000010;
typedef long long ll;
ll T, n, pri[maxn], cur, mu[maxn], sum_mu[maxn];
bool vis[maxn];
map<ll, ll> mp_mu;
ll S_mu(ll x) {
    if (x < maxn) return sum_mu[x];
    if (mp_mu[x]) return mp_mu[x];
    ll ret = 1ll;
    for (ll i = 2, j; i <= x; i = j + 1) {
        j = x / (x / i);
        ret -= S_mu(x / i) * (j - i + 1);
    }
    return mp_mu[x] = ret;
}
ll S_phi(ll x) {
    ll ret = 0ll;
    for (ll i = 1, j; i <= x; i = j + 1) {
        j = x / (x / i);
        ret += (S_mu(j) - S_mu(i - 1)) * (x / i) * (x / i);
    }
    return ((ret - 1) >> 1) + 1;
}
int main() {
    scanf("%lld", &T);
    mu[1] = 1;
    for (int i = 2; i < maxn; i++) {
        if (!vis[i]) {
            pri[++cur] = i;
            mu[i] = -1;
        }
        for (int j = 1; j <= cur && i * pri[j] < maxn; j++) {
            vis[i * pri[j]] = true;
            if (i % pri[j])
                mu[i * pri[j]] = -mu[i];
            else {
                mu[i * pri[j]] = 0;
                break;
            }
        }
    }
}

```



```

    a = a * a % P, m >>= 1;
}
return res;
}
void prime_work(int k) { // 线性筛 phi[]s
    bp[0] = bp[1] = 1, phi[1] = 1;
    for (int i = 2; i <= k; i++) {
        if (!bp[i]) p[++cnt] = i, phi[i] = i - 1;
        for (int j = 1; j <= cnt && i * p[j] <= k; j++) {
            bp[i * p[j]] = 1;
            if (i % p[j] == 0) {
                phi[i * p[j]] = phi[i] * p[j];
                break;
            } else
                phi[i * p[j]] = phi[i] * phi[p[j]];
        }
    }
    for (int i = 1; i <= k; i++)
        s[i] = (1ll * i * i % P * phi[i] % P + s[i - 1]) % P;
}
long long s3(long long k) {
    return k %= P, (k * (k + 1) / 2) % P * ((k * (k + 1) / 2) % P) % P;
} // 立方和
long long s2(long long k) {
    return k %= P, k * (k + 1) % P * (k * 2 + 1) % P * inv6 % P;
} // 平方和
long long calc(long long k) { // 计算 S(k)
    if (k <= pn) return s[k];
    if (s_map[k]) return s_map[k]; // 对于超过 pn 的用 map 离散存储
    long long res = s3(k), pre = 1, cur;
    for (long long i = 2, j; i <= k; i = j + 1)
        j = k / (k / i), cur = s2(j),
        res = (res - calc(k / i) * (cur - pre) % P) % P, pre = cur;
    return s_map[k] = (res + P) % P;
}
long long solve() {
    long long res = 0, pre = 0, cur;
    for (long long i = 1, j; i <= n; i = j + 1)
        j = n / (n / i), cur = calc(j),
        res = (res + (s3(n / i) * (cur - pre)) % P) % P, pre = cur;
    return (res + P) % P;
}
int main() {
    scanf("%lld%lld", &P, &n);
    inv2 = ksm(2, P - 2), inv6 = ksm(6, P - 2);
    pn = (long long)pow(n, 0.666667); // n^(2/3)
    prime_work(pn);
    printf("%lld", solve());
    return 0;
} // 不要为了省什么内存把数组开小..... 卡了好几次80

```

From:  
<https://wiki.cvbbacm.com/> - CVBB ACM Team

Permanent link:  
[https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal\\_string:lgwza:%E6%9D%9C%E6%95%99%E7%AD%9B&rev=1599448343](https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:lgwza:%E6%9D%9C%E6%95%99%E7%AD%9B&rev=1599448343) 

Last update: 2020/09/07 11:12