算法竞赛进阶 · 专题报告

莫比乌斯反演介绍

复旦大学 计算机科学技术学院

高庆麾 <19307130062>

2021 年 7 月 6 日

目录

第一部分　预备知识

一　函数定义

下面先介绍几个莫比乌斯反演中经常用到的函数定义。

(一)　莫比乌斯函数 $\mu(x)$

$$\mu(x)=\begin{cases}1, & x=1\\(-1)^n, & x=\prod_{i=1}^{n}p_i \quad (p_i\text{两两不同且为质数})\\0, & \exists p^2 \mid x \text{ 且}p\text{为质数}\end{cases}$$

可以认为，x 存在某个质因子，其指数大于 1 ，则 $\mu(x)$ 为 0 ，否则若有奇数个质因子，$\mu(x)=-1$ ，否则为 1

(二)　恒等函数 $id(x)$

$$id(x)\equiv x$$

(三)　单位函数 $e(x)$

$$e(x)=\begin{cases}1, & x=1\\0, & x>1\end{cases}$$

二　常用定理

下面是一些经常用到的定理。

(一)　gcd-lcm 定理

$$\gcd(i,j)*lcm(i,j)\equiv i*j$$

证明非常容易。

（二） 调和级数定理

$$\sum_{d=1}^{N} \frac{N}{d} = O(N \log N)$$

可以通过微积分证明。

（三） 平方根调和级数定理

$$\sum_{d=1}^{N} \sqrt{\frac{N}{d}} = N$$

可以通过微积分证明。

（四） Dirichlet 卷积运算律

Dirichlet 卷积满足交换律和结合律。

手动验证即可。

（五） Dirichlet 卷积积性性质

两个积性函数的 Dirichlet 卷积仍然是积性函数。

手动验证即可。

（六） Dirichlet 卷积计算复杂度

可以枚举每一个数字并对它的每一个倍数进行统计计算，这样做的复杂度是调和级数。

由上述调和级数定理，有 Dirichlet 卷积复杂度为 $\sum_{d=1}^{N} \frac{N}{d} = O(N \log N)$

（七） 莫比乌斯反演定理

有两种形式：

1. A 形式

$$F(n)=\sum_{d|n}f(d)\Leftrightarrow f(n)=\sum_{d|n}\mu(d)*F(\frac{n}{d})$$

2. B 形式

$$F(n)=\sum_{n|d}f(d)\Leftrightarrow f(n)=\sum_{n|d}\mu(\frac{d}{n})*F(d)$$

可以通过数学变换或 Dirichlet 卷积进行证明。

(八) φ 函数定理

定义 $F(x)$ 表示所有小于 x 且与 x 互质的数的和，其中 $x>1$ ，则有：

$$F(x)=\varphi(x)*\frac{x}{2}$$

考虑 $a<x$ ，若 $(a,x)=1$ ，则也有 $(x-a,x)=1$ ，由此容易证明。

第二部分　数论公式推导基本技术

一　μ 变换

这是最简单的第一步，因此有时容易被人忽略，形式如下：

$$[P(x)==k]\Leftrightarrow\sum_{d|P(x)/k}\mu(d)$$

这个公式也许刚开始会不太习惯，但如果学会并熟练运用这一技术，可以对莫比乌斯反演有更直观的认识与感受，进行更快速的推导而完全不需要套用莫比乌斯反演定理。

二　μ 提取

这往往是接续上一技术进行的，这个提取的含义就是把 $\mu(d)$ 的贡献系数计算出来，以便后续计算，形式如下：

$$\sum_{x}\sum_{d|P(x)/k}\mu(d)=\sum_{d}\mu(d)*num(d)$$

其中 $num(d)$ 表示 $\mu(d)$ 总共出现的次数。

最常用的一个形式是这样的：

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{d|\gcd(i,j)/k}\mu(d)=\sum_{d}\mu(d)*\left\lfloor\frac{n}{d*k}\right\rfloor*\left\lfloor\frac{m}{d*k}\right\rfloor$$

这里即有：

$$num(d)=\left\lfloor\frac{n}{d*k}\right\rfloor*\left\lfloor\frac{m}{d*k}\right\rfloor$$

三　φ 变换

上面介绍过这个式子，令 $Sum(x)$ 表示小于 x 且与 x 互质的数的和，则有：

$$Sum(x)=\frac{x*\varphi(x)}{2}$$

这是因为，如果 i 与 x 互质 $(i<x)$ ，那么易证 $x-i$ 也与 x 互质，这样，把它们两两组合相加，就得到上面的式子。为了计算方便，一般规定 $Sum(1)=1$ 。

四　积性函数提取

有时，算法的时间复杂度会因为需要枚举变量 k （或更多），同时在每个 k （或更多）之中进行一些分块前缀和计算而导致超时。

可以使用一些类似于“封装”的技巧，把公式的某一大部分的整体，看成一个数论函数（通常要求这个函数满足积性），然后对这个函数进行预处理，从而达到简化目的。

比如下面的式子：

$$\prod_{k=1}^{n}F(k)^{\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{d|\gcd(i,j)/k}\mu(d)}=\prod_{k=1}^{n}F(k)^{\sum_{d}\mu(d)*\left\lfloor\frac{n}{d*k}\right\rfloor*\left\lfloor\frac{m}{d*k}\right\rfloor}$$

它看似无法继续优化，但如果我们进行积性函数提取，则可以把前半部分进行整体代换，得到如下形式：

$$\prod_{T=1}^{n}\left(\prod_{d|T}F(d)^{\mu(\frac{T}{d})}\right)^{\left\lfloor\frac{n}{T}\right\rfloor*\left\lfloor\frac{m}{T}\right\rfloor}$$

这样，可以把前半部分提取出来，作为一个新的函数，形式如下：

$$G(T)=\prod_{d|T}F(d)^{\mu(\frac{T}{d})}$$

现在的任务就转变为预处理出所有的 $G(x)$ ，使原公式的计算效率有了显著提升。

而对于此类函数，如果它们是如下形式，就很好处理：

$$F(T)=\sum_{d|T}G(d)$$

或：

$$F(T)=\prod_{d|T}G(d)$$

且若 $G(x)$ 的计算复杂度为 $O(f)$ ，那么计算出所有 $F(x)$ 的复杂度即为 $O(f\cdot N\log N)$ 。

因为我们只要枚举 d ，然后对每一个 $d\times k$ 都进行更新即可。这是一个调和级数：

$$\sum_{d=1}^{N}\frac{N}{d}=O(N\log N)$$

而在上例中，计算 $G(x)$ 的复杂度为 $O(1)$ 的，所以预处理复杂度为 $O(N\log N)$ 。

再利用分块技术，可使计算复杂度由原先的 $O(QN)$ 优化为了 $O(N\log N+Q\sqrt{N})$ 。

五　Dirichlet 卷积

在推导公式的时候，往往可以加入一些处理，使公式的某一部分达到如下的形式：

$$\sum_{i|n}F(i)*G(\frac{n}{i})=(F*G)(n)$$

很明显，这就是 Dirichlet 卷积的形式，由此可以做如下两个方向的处理：

1. 直接计算这部分函数，根据 Dirichlet 卷积的计算复杂度定理，知其为 $O(N\log N)$ 。

2. 利用 Dirichlet 卷积的积性性质，两个积性函数的 Dirichlet 卷积仍然是积性函数。这样就可以使用线性筛进行 $O(N)$ 的预处理。

六　交换不变性

当一个式子中存在许多并列的 $\sum$ 或 $\prod$ 时，可以将它们任意调换顺序，同时转移相互依存的条件。这样即可达到一些继续推导和形式变换的目的。

比如，有如下计算式：

$$\textstyle\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{d=1}^{\min(n,m)}[\gcd(i,j)=d]=\sum_{d=1}^{\min(n,m)}\sum_{i=1}^{n}\sum_{j=1}^{m}\ [\gcd(i,j)=d]$$

同时，指数上的 $\sum$ 可以与外面的 $\prod$ 进行自由交换：

$$\textstyle\prod_{k=1}^{n}F(k)^{\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{d|\gcd(i,j)/k}\mu(d)}=\prod_{k=1}^{n}F(k)^{\sum_{d}\mu(d)*\left\lfloor\frac{n}{d*k}\right\rfloor*\left\lfloor\frac{m}{d*k}\right\rfloor}=\prod_{T=1}^{n}\left(\prod_{d|T}F(d)^{\mu(\frac{T}{d})}\right)^{\left\lfloor\frac{n}{T}\right\rfloor*\left\lfloor\frac{m}{T}\right\rfloor}$$

第三部分　数论公式计算基本技术

一　Dirichlet 卷积计算方法

前面已经多次提到，枚举每一个数字 x 并对它的每一个倍数 kx 进行计算即可。

二　积性函数线性筛法

对于一个积性函数，一般可以尝试一种方法，使得它可以在线性筛的过程被“顺便”计算出来，比如这个函数：

$$F(x)=\sum_{k|x}\mu(k)*k$$

可以证明这是一个积性函数。然后我们就可以尝试在线性筛中以 $O(N)$ 复杂度进行计算。

否则，就只能采用类似 Dirichlet 卷积的方法，在 $O(N\log N)$ 的时间下进行计算。这在 $N\approx 10^7$ 时很容易超时。

具体方法如下（在线性筛意义下给出）:

首先

$$F[1] = 1$$

其次

$$F(i * prime[j]) = \begin{cases} F[i] * (1 - prime[j]), & prime[j] \nmid i \\ F[i], & prime[j] \mid i \end{cases}$$

三　单维分块法计算

需要计算如下的式子:

$$\sum_{i=1}^{n} \left\lfloor \frac{n}{i} \right\rfloor$$

考虑不同的 $\left\lfloor \frac{n}{i} \right\rfloor$ 数量，可以对此进行分类讨论，对 $i < \sqrt{n}$ 的情况与 $i > \sqrt{n}$ 的情况分别观察，发现只有 $O(\sqrt{n})$ 的级别。

所以，我们希望找到这 $O(\sqrt{n})$ 个“关键点”，它们形成了一些分块，这样就不用在所有 n 个点上都统计一遍并加和，而是在这些关键点的分块上利用前缀和即可 $O(\sqrt{n})$ 地统计答案。

对于一个 i，它所在的分块的右端点有一个很简单的计算公式: $\left\lfloor \frac{n}{\left\lfloor \frac{n}{i} \right\rfloor} \right\rfloor$ ，这样就可以迅速遍历所有块。

四　双维分块法计算

我们需要计算如下的式子:

$$\sum_{i=1}^{n} \left\lfloor \frac{n}{i} \right\rfloor * \left\lfloor \frac{m}{i} \right\rfloor$$

可以继续沿用上面陈述的方法。不同的是，我们每次跳动的距离需要多考虑一个 m 。

也就是说，我们每次取对于 n 与 m 来说的分块右端点的较小值，然后跳到那里并统计这部分分块的答案贡献即可。

这相当于有最多 $O(\sqrt{n}+\sqrt{m})$ 个关键点插入进来，因此分块数也是这个级别。即复杂度为 $O(\sqrt{n}+\sqrt{m})$ 。

五 最大公约数统计

需要计算下面的式子：

$$\sum_{i=1}^{n}\sum_{j=1}^{m} gcd(i,j)$$

推导如下：

$$\begin{aligned}
&\textstyle\sum_{i=1}^{n}\sum_{j=1}^{m} gcd(i,j)\\
&=\textstyle\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{d=1}^{\min(n,m)} d*[gcd(i,j)=d]\\
&=\textstyle\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{d=1}^{\min(n,m)} d*\sum_{k|gcd(i,j)/d}\mu(k) \quad (\mu\text{变换})\\
&=\textstyle\sum_{d=1}^{\min(n,m)} d*\sum_{k=1}^{\min(n,m)}\mu(k)*\left\lfloor\frac{n}{d*k}\right\rfloor\left\lfloor\frac{m}{d*k}\right\rfloor \quad (\mu\text{提取})
\end{aligned}$$

此时固定 d ，对 k 使用分块计算的方法，能在下面的复杂度内求出答案：

$$O\left(\sum_{d=1}^{N}\sqrt{\frac{N}{d}}\right)$$

由平方根调和级数定理，这大约是 $O(N)$ 的级别。

但是，我们还可以进一步得到下面的式子：

$$\begin{aligned}
&\textstyle\sum_{d=1}^{\min(n,m)} d*\sum_{k=1}^{\max(n,m)}\mu(k)*\left\lfloor\frac{n}{d*k}\right\rfloor\left\lfloor\frac{m}{d*k}\right\rfloor\\
&=\textstyle\sum_{T=1}^{\min(n,m)}\left\lfloor\frac{n}{T}\right\rfloor\left\lfloor\frac{m}{T}\right\rfloor\sum_{d|T} d*\mu(\frac{T}{d}) \qquad (\text{积性函数提取})\\
&=\textstyle\sum_{T=1}^{\min(n,m)}\left\lfloor\frac{n}{T}\right\rfloor\left\lfloor\frac{m}{T}\right\rfloor(id*\mu)(T) \qquad (Dirichlet\text{卷积})\\
&=\textstyle\sum_{T=1}^{\min(n,m)}\left\lfloor\frac{n}{T}\right\rfloor\left\lfloor\frac{m}{T}\right\rfloor G(T)
\end{aligned}$$

其中使用了 Dirichlet 卷积。我们知道 Dirichlet 卷积计算复杂度是 $O(N\log N)$ ，所以可以先以 $O(N\log N)$ 的复杂度预处理出所有的 $G(x)$ ，然后计算出前缀和，再用双维分块的方法计算，复杂度是 $O(\sqrt{n}+\sqrt{m})$ 的。

六　最小公倍数统计

需要计算下面的式子：

$$\sum_{i=1}^{n}\sum_{j=1}^{m} lcm(i,j)$$

首先根据常用定理，有：

$$lcm(i,j) = \frac{i*j}{gcd(i,j)}$$

所以式子就化为了：

$$\sum_{i=1}^{n}\sum_{j=1}^{m} \frac{i*j}{gcd(i,j)}$$

下面就对这个式子进行详细的推导。可以发现，此时分母起到很大的作用。两项如果分母不同，很难把它们归到一起。所以很自然地想到枚举 gcd 的取值。

推导如下：

$$\begin{aligned}
&\textstyle\sum_{i=1}^{n}\sum_{j=1}^{m}\frac{i*j}{gcd(i,j)}\\
&\textstyle= \sum_{d=1}^{\min(n,m)}\sum_{i=1}^{n}\sum_{j=1}^{m}\frac{i*j}{d}*[gcd(i,j)=d]\\
&\textstyle= \sum_{d=1}^{\min(n,m)}\sum_{i'=1}^{n}\sum_{j'=1}^{m} i'*j'*d*[gcd(i',j')=1]\\
&\textstyle= \sum_{d=1}^{\min(n,m)}\sum_{i'=1}^{n}\sum_{j'=1}^{m}\sum_{k|gcd(i',j')} i'*j'*d*\mu(k) \quad (\mu\text{变换})\\
&\textstyle= \sum_{d=1}^{\min(n,m)} d*\sum_{k}\mu(k)*Sum(\lfloor\frac{n}{k*d}\rfloor,\lfloor\frac{m}{k*d}\rfloor)*k^2 \qquad (\mu\text{提取})
\end{aligned}$$

其中

$$Sum(a,b) = \tfrac{a*(a+1)}{2} * \tfrac{b*(b+1)}{2}$$

此时，式子似乎不能再优化。于是继续使用以前的技术，做积性函数提取和变量归一（把变量 k,d 归一到 T），可以继续得到下面的式子：

$$\begin{aligned}
&\sum_{d=1}^{\min(n,m)} d * \sum_k \mu(k) * Sum(\lfloor \tfrac{n}{k*d} \rfloor, \lfloor \tfrac{m}{k*d} \rfloor) * k^2 && (\mu\text{提取})\\
&= \sum_{T=1}^{\min(n,m)} T * Sum(\lfloor \tfrac{n}{T} \rfloor, \lfloor \tfrac{m}{T} \rfloor) * \sum_{k|T} \mu(k) * k && (\text{变量归一化})\\
&= \sum_{T=1}^{\min(n,m)} T * Sum(\lfloor \tfrac{n}{T} \rfloor, \lfloor \tfrac{m}{T} \rfloor) * F(T) && (\text{积性函数提取})
\end{aligned}$$

这里解释一下为什么 $F(T)$ 是积性函数。显然 $F(T)$ 是两个积性函数的狄利克雷卷积（ $F = f * 1$，且其中 $f = \mu(k) * k$，显然为积性函数)，所以 F 是一个积性函数。

那么，对于积性函数，我们可以怎么做呢?

一开始想要使用杜教筛，后来发现好像不用这么麻烦，只要能列出该函数在线性筛意义下的转移，就可以在线性筛的同时“顺便”计算该函数。

首先有 $F[1] = 1$ ，然后对两种情况分别计算：

$$F(i * prime[j]) = \begin{cases} F[i] * (1 - prime[j]), & prime[j] \nmid i \\ F[i], & prime[j] \mid i \end{cases}$$

这样，用线性筛就可以得到所有的 F 。然后，使用双维分块法技术就可以进行计算了。总复杂度为 $O(n + Q * (\sqrt{n} + \sqrt{m}))$ 。

第四部分　例题分析

一　T1. [CQOI2007] 余数之和 sum

(一)　题目描述

给出正整数 n 和 k，计算 j(n, k)=k mod 1 + k mod 2 + k mod 3 + ⋯ + k mod n 的值，其中 k mod i 表示 k 除以 i 的余数。例如 j(5, 3)=3 mod 1 + 3 mod 2 + 3 mod 3 + 3 mod 4 + 3 mod 5=0+1+0+3+3=7

其中 50% 的数据满足：$1 \leqslant n, k \leqslant 1000$，100% 的数据满足：$1 \leqslant n, k \leqslant 10^9$

(二)　思路分析

显然，$x \bmod y = x - y\lfloor \frac{x}{y} \rfloor$，所以有：

$$\sum_{i=1}^{n}(k \bmod i) = \sum_{i=1}^{n}\left(k - i\left\lfloor\frac{k}{i}\right\rfloor\right) = nk - \sum_{i=1}^{n} i\left\lfloor\frac{k}{i}\right\rfloor$$

对于后面的和式部分，可以直接使用单维分块，然后计算出一个 $\left\lfloor\frac{k}{i}\right\rfloor$ 相同的块中，i 的和是多少，这显然很容易计算，只要知道这个块的左右端点即可。

由此复杂度为 $O(\sqrt{k})$ 。

本题实际上和莫比乌斯反演没什么关系，但可以体现出它里面一些技巧在实际中的应用方法。

（三） 解题代码

```
#include <cstdlib>
#include <cstdio>
#include <algorithm>
#define LL long long int
using namespace std;
void work(LL n, LL k){
    LL ans = 0;
    LL con = min(n, k);
    for (LL i = 1; i <= con; i++){
        LL next = k / (k / i);
        next = min(next, con);
        LL d = k / i;
        LL a = k % i;
        //LL num = a / d;
        /*if (num < next - i){
            ans += (a + a - num * d) * (num + 1) >> 1;
            a = a - (num + 1) * d + i + num + 1;
            ans += (a + a - d * (next - i - num)) * (next - i - num) >> 1;
        }
        else {
            num = next - i;
            ans += (a + a - num * d) * (num + 1) >> 1;
        }*/
        LL num = next - i;
        ans += (a + a - num * d) * (num + 1) >> 1;
        i = next;
    }
    if (n > k){
```

```
        ans += (n - k) * k;
    }
    printf("%lld", ans);
}
int main(){
    freopen("rest.in", "r", stdin);
    freopen("rest.out", "w", stdout);
    LL n, k;
    scanf("%lld %lld", &n, &k);
    work(n, k);
    return 0;
}
```

二 T2. Spoj5971 LCM Sum

(一) 题目描述

Given n, calculate the sum LCM(1,n) + LCM(2,n) + .. + LCM(n,n), where LCM(i,n) denotes the Least Common Multiple of the integers i and n.

$1 \leqslant T \leqslant 300000, 1 \leqslant n \leqslant 1000000$

(二) 思路分析

实际上，上文讲述的内容足够应付绝大多数此类题目。本题只是上文中最小公倍数统计的一个特例版本。但本题可以用到上文一个没怎么提及的技术。

公式推导如下：

$$\begin{aligned} Ans &= \sum_{i=1}^{n} lcm(i,n) \\ &= \sum_{i=1}^{n} \frac{n*i}{gcd(i,n)} \\ &= n * \sum_{d|n} \sum_{i=1}^{\frac{n}{d}} i * [gcd(i, \tfrac{n}{d}) = 1] \end{aligned}$$

下面当然可以像之前推最小公倍数统计时一样处理。

但实际上，可以发现后面那个和式实际是在统计小于 $\frac{n}{d}$ 且与之互质的数的和。这就可以想到上面介绍的 φ 变换（虽然用的很少）。

继续推导，有：

$$n * \sum_{d|n} \sum_{i=1}^{\frac{n}{d}} i * [gcd(i, \tfrac{n}{d}) = 1]$$
$$= n * \sum_{d|n} \frac{\frac{n}{d} * \varphi(\frac{n}{d})}{2} \qquad (\varphi\text{变换})$$
$$= n * G(n)$$

(三) 解题代码

```
#include <cstdlib>
#include <cstdio>
#include <algorithm>
#define LL long long int
#define maxn 1000005
using namespace std;

LL fi[maxn];
LL g[maxn];
LL prime[maxn], cnt;
bool vis[maxn];

void init(){
    fi[1] = 1;
    for (LL i = 2; i < maxn; i++){
        if (!vis[i]){
            prime[cnt++] = i;
            fi[i] = i - 1;
        }
        for (LL j = 0; j < cnt; j++){
            if (prime[j] * i >= maxn) break;
            vis[prime[j] * i] = 1;
            fi[prime[j] * i] = fi[i] * (prime[j] - 1);
            if (i % prime[j] == 0){
                fi[prime[j] * i] = fi[i] * prime[j];
                break;
            }
        }
    }
    for (LL i = 1; i < maxn; i++) g[i] = 1;
    for (LL i = 2; i < maxn; i++){
        LL add = (i * fi[i]) >> 1;
        for (LL j = 1; j * i < maxn; j++){
```

```
                g[i * j] += add;
            }
    }
    for (LL i = 1; i < maxn; i++){
        g[i] *= i;
    }
}

int main(){
    init();
    freopen("LCM.in", "r", stdin);
    freopen("LCM.out", "w", stdout);
    LL T, n;
    scanf("%lld", &T);
    while (T--){
        scanf("%lld", &n);
        printf("%lld\n", g[n]);
    }
    return 0;
}
```

第五部分　后记

上面所述的技术，应该可以解决几乎所有莫比乌斯反演问题。只要再更加灵活地练习和掌握这些技术，公式推导题目其实都很简单。