关于积性函数求和的一些探讨

周康阳

杭州学军中学教育集团文渊中学

Feburary 2nd, 2024

前言

积性函数求和是一类重要的数论问题。关于该问题，在此前的研究旨在减少 soft-$\mathcal{O}(n^{2/3})$ 中的 $\log$ 因子数量[1]。
而我在思考后发现该问题可以做到 $\mathcal{O}(n^{1/2}\operatorname{poly}\log n)$ 的时间复杂度 [2]。
本文将介绍这一筛法。

[1]一个相关的研究是 https://negiizhao.blog.uoj.ac/blog/8961
[2]初稿于 2023-07-11 首发于 https://www.cnblogs.com/zkyJuruo/p/17544928.html

Contents

定义与约定

Definition

数论函数是定义域为正整数集，陪域为复数域的函数。

对于一个数论函数 f，我们约定 $S_f(n) = \sum_{i=1}^{n} f(i)$。

Definition

积性函数是指一个定义域为正整数 n 的数论函数 $f(n)$，且满足如下性质：$f(1) = 1$，且当 a 和 b 互质时，$f(ab) = f(a)f(b)$。

对于一个积性函数 f，只要知道了对于所有 $p \in \mathbb{P}, k \in \mathbb{Z}^+$ 的 $f(p^k)$，我们就可以知道整个 f 函数。

定义与约定

一些常见的积性函数在 $p \in \mathbb{P}, k \in \mathbb{Z}^+$ 的取值如下：

- ϵ 函数（单位函数），满足 $\epsilon(p^k) = 0$。
- μ 函数，满足 $\mu(p^k) = -[k = 1]$。
- φ 函数，满足 $\varphi(p^k) = (p - 1)p^{k-1}$。
- I 函数，满足 $I(p^k) = 1$。
- id_t 函数，满足 $id_t(p^k) = p^{kt}$。

Dirichlet 卷积

Definition

对于数论函数 f 和 g，定义其 *Dirichlet* 卷积

$$(f * g)(x) = \sum_{d|x} f(d)g(\frac{x}{d})$$

如果将 *Dirichlet* 卷积视为数论函数的乘法，函数的直接加和为数论函数的加法。那么数论函数的加法和乘法是满足交换律、结合律和分配律的。

Powerful Number

Definition

Powerful Number 是指一个正整数 m，满足对于任意 m 的质因数 p，均有 $p^2|m$。

接下来，我们用 PN 简称 Powerful Number。在公式中，用 PN 代表所有 Powerful Number 构成的集合。

Theorem

对于所有 PN，我们都可以将其表示成 a^2b^3 的形式（其中 a 和 b 都是正整数）。

对于一个 PN 数 $m=\prod_{i=1}^{k}p_i^{\alpha_i}$，对每个 $p_i^{\alpha_i}$ 都构造出 a^2b^3 的即可。这个只要把 α_i 拆成 $2x+3y$ 的形式就行了。

Powerful Number

Theorem

n 以内的 *PN* 个数为 $\mathcal{O}(\sqrt{n})$。

Proof.

把所有 n 以内的 PN 表示成 a^2b^3 的形式。枚举 a，我们就可以得到 PN 数不会超过 $\sum_{a=1}^{\sqrt{n}}(\frac{n}{a^2})^{1/3}$。这个可以估计为

$$\begin{aligned}\int_1^{\sqrt{n}}(\frac{n}{x^2})^{1/3}\,\mathrm{d}x &= n^{1/3}\int_1^{\sqrt{n}}x^{-2/3}\,\mathrm{d}x\\ &= n^{1/3}(3n^{1/6}-3)\end{aligned}$$

即 $\mathcal{O}(\sqrt{n})$ 级别。 □

可以通过深搜在 $\mathcal{O}(\sqrt{n})$ 内找出所有 n 以内的 PN。

Powerful Number

下面看一道例题。

Problem

(UOJ Long Round 1 校验码的 *n=1* 部分分*)* 给定 c。积性函数函数 $q(x)$ 满足 $q(p^k)=p^{c\lfloor\frac{k}{2}\rfloor}$。求 $\sum_{i=1}^{m} q(i)$，答案对 2^{32} 取模。$n \le 1.2\times 10^{11}$

Powerful Number

设计积性函数 $f(x)$ 满足 $f(1)=1$，此时 $f(p)=q(p)$。然后构造积性函数 g 满足 $g*f=q$。
对于素数 p，有 $g(p)f(1)+g(1)f(p)=q(p)$，解得 $g(p)=0$。因此 g 函数只在 PN 处有值!

$$\sum_{i=1}^{m} q(i)=\sum_{i=1}^{m}\sum_{jk=i} g(j)f(k)$$

$$=\sum_{j\in PN, j\le m} g(j)\sum_{k=1}^{\lfloor\frac{m}{j}\rfloor} f(k)$$

$$=\sum_{j\in PN, j\le m} g(j)\lfloor\frac{m}{j}\rfloor$$

Powerful Number

对于一个质数 p，我们有 $h(p^k)=\sum_{i=0}^{k} f(p^i)g(p^{k-i})$。
因此 $g(p^k)=h(p^k)-\sum_{i=1}^{k} f(p^i)g(p^{k-i})$，我们可以按 k 从小到大的顺序解出 $g(p^k)$。
在 DFS 找 PN 的同时维护 g 函数值。这样，这个问题就可以 $\mathcal{O}(\sqrt{m})$ 解决了。

杜教筛

对于一些特殊的积性函数（如 μ 和 ϕ），我们可以通过构造 $Dirichlet$ 卷积在 $\mathcal{O}(n^{\frac{2}{3}})$ 的时间复杂度计算其前缀和。这种算法在国内信息学竞赛界被称为"杜教筛"。

对于 μ 函数，我们有 $\mu * I = \epsilon$。

不妨考虑更一般的问题。如果有数论函数 A, B, C 满足 $A(1) = B(1) = C(1) = 1$ 且 $A * B = C$，并且我们能快求出 S_B 和 S_C 的点值，我们该如何求出 $S_A(n)$ 呢？

杜教筛

注意到

$$\begin{aligned}\sum_{i=1}^{n} C(i) &= \sum_{i=1}^{n} \sum_{jk=i} A(j)B(k) \\ &= \sum_{k=1}^{n} B(k) \sum_{j=1}^{\lfloor \frac{n}{k} \rfloor} A(j) \\ &= S_A(n) + \sum_{k=2}^{n} B(k) S_A(\lfloor \frac{n}{k} \rfloor)\end{aligned}$$

因此 $S_A(n) = S_C(n) - \sum_{k=2}^{n} B(k) S_A(\lfloor \frac{n}{k} \rfloor)$，可以递归求解 $S_A(n)$。

杜教筛

注意到 $\lfloor\frac{\lfloor\frac{n}{a}\rfloor}{b}\rfloor = \lfloor\frac{n}{ab}\rfloor$，在递归过程中，需要被计算的实际上只有 $S_A(\lfloor\frac{n}{k}\rfloor)$（$k$ 是正整数）。

Theorem

对于正整数 n，集合 $\{\lfloor\frac{n}{i}\rfloor | i \in \mathbb{Z}^+\}$ 的大小是 $\mathcal{O}(\sqrt{n})$ 的。

Proof.

对于 $k \le \sqrt{n}$，只有 $\sqrt{n}$ 个 $\lfloor\frac{n}{k}\rfloor$，显然只有 $\mathcal{O}(\sqrt{n})$ 种；
对于 $k > \sqrt{n}$，$\lfloor\frac{n}{k}\rfloor \le \frac{n}{\sqrt{n}}$，因此也只有 $\mathcal{O}(\sqrt{n})$ 种。 □

因此，我们只需要计算 S_A 的 $\mathcal{O}(\sqrt{n})$ 个点值。

杜教筛

在计算单个 $S_A(m)$ 时，将连续的相同的 $\lfloor \frac{m}{k} \rfloor$ 一起计算（即整除分块），就能在 $\mathcal{O}(\sqrt{m})$ 的复杂度内递归到子问题了。

对于 $m \le n^{\frac{2}{3}}$ 的 $S_A(m)$，我们通过提前预处理出 A 的前 $n^{\frac{2}{3}}$ 个点值来解决。对于 $m > n^{\frac{2}{3}}$，使用上面的递归式计算，这部分的时间复杂度 $\mathcal{O}(\sum_{i=1}^{n^{\frac{1}{3}}} \sqrt{\frac{n}{i}})$。这个可以估算为 $\mathcal{O}(\int_1^{n^{\frac{1}{3}}} \sqrt{\frac{n}{i}}) = \mathcal{O}(n^{\frac{2}{3}})$。

总复杂度即为 $\mathcal{O}(n^{\frac{2}{3}})$ 加上预处理出 $\mathcal{O}(n^{\frac{2}{3}})$ 预处理出 A 的前 $n^{\frac{2}{3}}$ 个点值的复杂度（μ 和 φ 都是可以 $\mathcal{O}(n^{\frac{2}{3}})$ 预处理的）。

杜教筛

Definition

定义数论函数 f 关于 n 的块筛为 $S_f(n)$ 在 $\{\lfloor \frac{n}{i} \rfloor | i \in \mathbb{Z}^+\}$ 的点值。

在上述过程当中，我们只用了 B 和 C 关于 n 的块筛，就求出了 A 关于 n 的块筛。

Contents

块筛卷积

对于数论函数 A, B, C 满足 $A(1) = B(1) = C(1) = 1$ 且 $A * B = C$，上文的杜教筛根据 B 和 C 关于 n 的块筛解出了 A 关于 n 的块筛。
在这里，我们先考虑一个更加简单的问题：已知数论函数 A 和 B 关于 n 的块筛，要求求出满足 $C = A * B$ 关于 n 的块筛。我们将 C 称为 A 和 B 的块筛卷积。块筛上的运算仍然具有交换律，结合律和分配律。
这个问题实际上是可以 $\mathcal{O}(n^{\frac{1}{2}} \operatorname{poly} \log n)$ 解决的。

块筛卷积

下文中，我们默认块筛是关于 n 的块筛。
我们要解决的是 $C(z)=\sum_{xy=z}A(x)B(y)$ 在所有 $\lfloor \frac{n}{t} \rfloor$ 上的前缀和。
首先考虑 $x>\sqrt{n}$ 的情况（$y>\sqrt{n}$ 是对称的）。如果 $xy \le \lfloor \frac{n}{t} \rfloor$，那么 $x \le \frac{n}{ty}$。所以枚举 t 和 y，有贡献的 x 是一段前缀。这部分的复杂度是 $\mathcal{O}(\sqrt{n}\log n)$。
这样我们就只需要解决 $x \le \sqrt{n}$ 且 $y \le \sqrt{n}$ 的情况了。

块筛卷积

$xy \le \frac{n}{t}$ 等价于 $\ln(x) + \ln(y) \le \ln(\frac{n}{t})$。
我们做一个估计：选取一个块长 $S(3 \le S \le n)$，我们认为 $\ln(x) + \ln(y) \le \ln(\frac{n}{t})$ 当且仅当 $\lceil S\ln(x)\rceil + \lceil S\ln(y)\rceil \le S\ln(\frac{n}{t})$。
这个估计可以用多项式乘法完成。设计多项式 $F(z), G(z)$，其中 $[z^k]F(z)$ 为满足 $\lceil S\ln(x)\rceil = k$ 的 $A(x)$ 的和，$[z^k]G(z)$ 为满足 $\lceil S\ln(x)\rceil = k$ 的 $B(x)$ 的和。计算多项式 $H(z) = F(z)G(z)$，则 $[z^k]H(z)$ 就是 $\lceil S\ln(x)\rceil + \lceil S\ln(y)\rceil = k$ 的 $A(x)B(y)$ 和。使用一次长度为 $S\ln(n)$ 的卷积即可。

块筛卷积

但是这样做显然是不对的。观察我们在什么情况下会估计错：只有 $\ln(x)+\ln(y)\le\ln(\frac{n}{t})$ 且 $\lceil S\ln(x)\rceil+\lceil S\ln(y)\rceil > S\ln(\frac{n}{t})$ 时才会估错。此时有 $S\ln(x)+S\ln(y)\in(S\ln(\frac{n}{t})-2, S\ln(\frac{n}{t})]$。
注意到这个时候 x,y,t 必须满足 $\ln(x)+\ln(y)+\ln(t)\in(\ln(n)-\frac{2}{S},\ln(n)]$，即 $xyt\in(ne^{-\frac{2}{S}},n]$。其中 $e^{-\frac{2}{S}}$ 是 $1-\mathcal{O}(\frac{1}{S})$ 级别的。
因此，xyt 是在一个长度为 $\mathcal{O}(\frac{n}{S})$ 的区间内的！

块筛卷积

不妨假设这个区间是 $[n-L,n]$。我们使用区间筛得到这个区间内所有数的质因数分解。更具体地说，先筛出所有不超过 $\sqrt{n}$ 的小质数，对于每个质数，我们都可以求得区间 $[n-L,n]$ 中被它整除的数。这样我们就筛出了 $[n-L,n]$ 的所有数的 $\le \sqrt{n}$ 的质因子。而 $>\sqrt{n}$ 的质因子最多只有一个，所以只要看这个数除掉所有小质数后的值就行了。

得到质因数分解后，我们就可以通过在质因子上 DFS 快速找出所有可能的 (x,y,t) 了，对于一个 (x,y,t) 可以 $\mathcal{O}(1)$ 检查它是否被正确地估计了。对于一个 $v\in[n-L,n]$，它贡献时间复杂度是 $d_3(v)$（其中 $d_3(v)=\sum_{xyz=v}1$）。

块筛卷积

而根据解析数论中的结论[3]，$\sum_{i\le n} d_3(i) = nP_3(\ln n) + O(n^{43/96+\epsilon})$。所以 $\sum_{i=n-L}^{n} d_3(i) = LP_3(\ln n) + \mathcal{O}(n^{43/96+\epsilon})$，其中 P_3 是二次多项式。算法的输出已经有 $\mathcal{O}(\sqrt{n})$，比 $\mathcal{O}(n^{43/96+\epsilon})$ 大，所以可以忽略这部分的复杂度。剩下部分的复杂度就是 $\mathcal{O}(L\log^2 n)$，即 $\mathcal{O}(\frac{n}{S}\log^2 n)$。

这么做的总复杂度是 $\mathcal{O}(S\log^2 n + \frac{n}{S}\log^2 n)$，取 $S = \sqrt{n}$ 即可得到 $\mathcal{O}(\sqrt{n}\log^2 n)$ 的时间复杂度。

[3] https://en.wikipedia.org/wiki/Divisor_summatory_function

Contents

积性函数求和

对于一个满足一些特殊性质（比如满足 $f(p^k) = p^k(p^k - 1)$ 的积性函数）的积性函数 f，该如何求 f 的块筛呢？
首先，$f(p^k)(k \geq 2)$ 的值并不重要。如果另一个积性函数 g 满足 $f(p) = g(p)$，且 g 的前缀和容易计算，那么我们只需要设计满足 $h * g = f$ 的积性函数 h，根据前文的结论，h 只在 PN 处有值且这些位置上的值可以 $\mathcal{O}(\sqrt{n})$ 算出。因此容易 $\mathcal{O}(\sqrt{n})$ 算出 h 的块筛，然后通过 h 和 g 的块筛卷积算出 f 的块筛。

积性函数求和

我们先设计数论函数 q 满足 q 只在质数处的值非零，且 $q(p)=f(p)$。

定义一个数论函数 q（$q(1)=0$）的 $\exp(q)$ 为 $\sum_{i=0}\frac{q^i}{i!}$，其中卷积是块筛卷积。观察 $d=\exp(q)$。对于一个数 $x=\prod_{i=1}^{k}p_i^{\alpha_i}$，$d(x)$ 只有在 $q^{\sum_{i=1}^{k}\alpha_i}$ 项上有值，且值为 $\frac{\prod_{i=1}^{k}q(p_i)^{\alpha_i}}{(\sum_{i=1}^{k}\alpha_i)!}\binom{\sum_{i=1}^{k}\alpha_i}{\alpha_1,\alpha_2,\dots,\alpha_k}=\frac{\prod_{i=1}^{k}q(p_i)^{\alpha_i}}{\prod_{i=1}^{k}\alpha_i!}$。

因此，d 函数是一个积性函数，且 $d(p^k)=\frac{f(p)^k}{k!}$，满足 $d(p)=f(p)$。这样，如果我们能求出 q 的块筛，我们就可以通过 $\log n$ 次（在 $i>\log n$ 时 q 函数的前 n 项没有值）块筛卷积得到 d 函数的块筛，从而得到 f 的块筛。

积性函数求和

怎么得到 q 的块筛呢？考虑一个类似 Min_25 筛的想法。将 q 函数拆成若干个 q_i 的带权和，并通过这些 q_i 的块筛来算出 q 的块筛。比如 $q(p) = p(p-1)$，我们就可以拆成 $q_1(p) = p^2$，$q_2(p) = p$，然后就有 $q = q_1 - q_2$。
当 $q(p)$ 是关于 p 的常数次多项式时，我们都可以按照这种方法拆函数。

积性函数求和

如何求出 q_i 的块筛呢？这里我们反过来。
以求 $q_1(p)=p^2$ 为例。对于一个满足 $d(p^k)=q_1(p)^k$ 的积性函数 d，它满足 $d(x)=x^2$，块筛很好算；而对于满足 $d(p^k)=\frac{q_1(p)^k}{k!}$ 的积性函数 d，由于它和上个函数在 p 处的点值相同，所以也可以用 PN 在一次块筛卷积的复杂度内相互转化，所以它的块筛也是可以算出的。

积性函数求和

而满足 $d(p^k)=\frac{q_1(p)^k}{k!}$ 的函数 d，正是 $\exp(q_1)$。这启发我们用一个类似 $\ln$ 的思路来从 d 反推出 q_1。
定义数论函数 d（$d(1)=1$）的 $\ln(d)$ 为 $\sum_{i=1}\frac{(-1)^{i-1}(d-\epsilon)^i}{i}$。
我们有 $q_1=\ln(d)$，证明如下。

积性函数求和

我们只要分别说明 $\exp(\ln(d)) = d$ 和满足 $\exp(q) = d$ 的 q 是唯一的即可。

如果满足 $\exp(q) = d$，那么 $\epsilon + q + \sum_{i\geq 2} \frac{q^i}{i!} = d$。我们可以从小到大依次确定 $q(i)$ 的值：设函数 $p = \sum_{i\geq 2} \frac{q^i}{i!}$，则 $p(i)$ 只和 $q(j)(j < i)$ 有关，所以可以用 $q(i) = d(i) - [i = 1] - p(i)$ 解出 $q(i)$。因此 q 函数是可以被唯一确定的。

积性函数求和

然后说明 $\exp(\ln(d)) = d$。

$$\exp(\ln(d)) = \sum_{j=0} \frac{\left(\sum_{i=1} \frac{(-1)^{i-1}(d-\epsilon)^i}{i}\right)^j}{j!}$$

使用块筛的分配律和结合律将其展开，我们将得到一个 $\sum_i a_i d^i$ 形式的式子。不妨用多项式 $A(z)$ 来刻画它，其中 $[z^k]A(z) = a_k$。而 $\exp$ 和 $\ln$ 在块筛卷积中的定义和多项式相同（即 $\exp(z) = \sum_{i=0} \frac{z^i}{i!}$，$\ln(1+z) = \sum_{i=1} \frac{(-1)^{i-1} z^i}{i}$），$A(z) = \exp(\ln(z)) = z$，上面的 $a_i = [i = 1]$，因此 $\exp(\ln(d)) = d$。

积性函数求和

上述的 $\ln$ 中也只需要枚举 $i \le \log n$，所以也只需要 $\log n$ 次块筛卷积。
因此，该算法的时间复杂度是 $\mathcal{O}(\sqrt{n}\log^3 n)$。

Contents

总结

在本文中，我们得出了一种比较具有拓展性的筛法，将块筛卷积、积性函数块筛前缀和和质数前缀统计问题做到了 $\mathcal{O}(\sqrt{n}\,\mathrm{polylog}\,n)$ 的时间复杂度（后两者需要满足 $f(p)$ 是关于 p 的常数次多项式或一些其他的特殊性质）。

在我的集训队论文中，还提到了对这个算法的一些优化，并且目前的时间复杂度是 $\mathcal{O}(\frac{\sqrt{n}\log^2 n}{\sqrt{\log\log n}})$，而且很可能可以被进一步优化。

欢迎大家来和我探讨这个问题。

致谢

感谢中国计算机学会提供学习和交流的平台。
感谢国家集训队教练彭思进、杨耀良的指导。
感谢学军中学徐先友等老师的关心与指导。
感谢家人对我的陪伴与支持。
感谢信友队的同学、学长给予我帮助与陪伴。
感谢章弥炫、张恒毅等同学为本文验稿并提出建议。
感谢其他给予我帮助的老师与同学。
感谢大家的聆听。