简单易懂的质数筛法

过时数论魔法

陈牧歌

北京大学

2018 年 5 月 13 日

前言

- Q: 你是谁？没听过呀
- A: 半吊子大三生，没打过 APIO，CTSC 也没拿过奖的不知名前 OI 选手
- Q: 你要讲什么？
- A: 小学必修的质数筛法的一点点拓展呢
- Q: 为啥讲这个呢？
- A: 过气选手只会这个了，惨兮兮。

内容安排

内容比较简单轻松，希望大家不要掉线

- Eratosthenes 筛法介绍
- “杜教筛”与“洲阁筛”科普
- yet another 扩展 Eratosthenes 筛

其实参加过 CTSC 的同学可能已经发现了，本次讲课与朱震霆同学的《一些特殊的数论函数求和问题》大量重合

什么是 Eratosthenes 筛

- Eratosthenes 筛是人类历史记录下最古算法之一
- 用于找出一定范围内 ($[1,n]$) 的所有质数
- 根据定义，任一合数都是某个大于 1 的数的倍数
- 从 2 开始，将每个数的各个倍数，标记成合数
- 范围内所有未被标记的数，就是所求的全部素数

```
for (int i = 2; i <= n; i++)
  isPrime[i] = true;
for (int i = 2; i <= n; i++)
  for (int j = i+i; j <= n; j+=i)
    isPrime[j] = false;
^^I
```

简单优化 # 1

- 任一合数都是某个质数的倍数
- 所以只需要用质数去筛即可。

```
for (int i = 2; i <= n; i++)
  isPrime[i] = true;
for (int i = 2; i <= n; i++)
  if (isPrime[i]) {
    for (int j = i+i; j <= n; j+=i)
     isPrime[j] = false;
  }
^^I
```

复杂度是多少?

简单优化 # 1 续

- 容易知道复杂度就是 $\sum_{p\le n}\lfloor \frac{n}{p}\rfloor$，其中 p 是质数
- 问题是怎么简化求和式，如果具有一定的微积分知识，就能证明 $\sum_{p\le n}\lfloor \frac{n}{p}\rfloor = \Theta(n\log\log n)$
- 即 Mertens 第二定理。
- 证明后面有时间再证。反正今天复杂度证明不差这一个（笑）。

简单优化 # 2

- 事实上一个 n 以内的数若是合数，必然有不大于 $\sqrt{n}$ 的约数
- 且若发现一个质数 p，也不应该从 $2*p$ 开始筛起，应该从 $p*p$ 筛起，因为更小的合数已经被别的质数筛掉了。

```
for (int i = 2; i <= n; i++)
  isPrime[i] = true;
for (int i = 2; i*i <= n; i++)
  if (isPrime[i]) {
    for (int j = i*i; j <= n; j+=i)
      isPrime[j] = false;
  }
^^I
```

继续优化可以导出所谓的“线形筛”

SPOJ PRIME1[1]

- 输出在区间 $[m, n]$ 内的所有质数
- $1 \leq m \leq n \leq 10^9$
- $n - m \leq 100000$

[1] http://www.spoj.com/problems/PRIME1/

SPOJ PRIME1

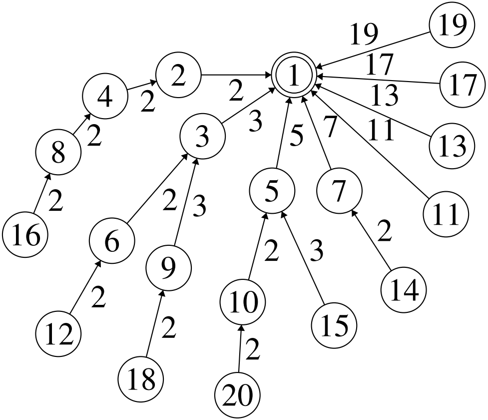
- 暴力筛出 $[1,n]$ 内的所有质数显然时空都不可接受
- 注意到 $n-m \leq 100000$ 这一条件，区间很小
- 预处理出 $\sqrt{n}$ 以内的所有质数，将它们在 $[m,n]$ 里的倍数都划去，剩下的就是质数
- 时间复杂度为 $\sum_{p \leq \sqrt{n}} \frac{100000}{p}$，可以通过此题

Euler 筛（线性筛）

- 之前的筛法，一个合数会被筛去多次（被它的每个质因数筛一次）
- 如果我们能保证每个合数只被筛一次，就能得到时间复杂度为线性的筛法。
- Euler 筛就是这样的筛法，它保证每个合数只被其最小质因数筛去。

```
for (int i = 2; i <= n; i++) {
  if (isPrime[i]) prime.push_back(i);
  for (int j = 0; j < prime.size(); j++) {
    if (i * prime[j] > N) break;
    isPrime[i * prime[j]] = false;
    if (i % prime[j] == 0) break;
  }
}
^^I
```

正确性与复杂度证明



- 将 $[1,n]$ 中的每个数抽象成一个图的节点，我们可以建立一棵有根树。
- 对于每个数 u，找出最小的质数 p，使得 $p|u$。
- 设定 u/p 为 u 的父节点
- 容易发现对于 Euler 筛实际上就是枚举每个节点，标记其直接子节点为合数
- 复杂度立即得证为 $O(n)$

筛法与积性函数

- 以排除合数的思路求质数，线性即为最优复杂度了。
- 因为合数的数目也是 $O(n)$ 的，全部筛去至少要 $O(n)$ 的时间。
- 但是筛法不仅能用来求素数，还能用来求一个积性函数给定范围内的所有值。

积性函数

- 对于定义域在 $\mathbb{N}^+$ 上的函数 f
- 若满足对于任意互质正整数对 (a,b) 均有 $f(a\times b)=f(a)\times f(b)$
- 则称 f 为积性函数
- 对于任意大于 1 的整数 N, 设 $N=\prod p_i^{q_i}$, 其中 p_i 为互不相同的质数。
- 那么容易得到 $f(N)=f(\prod p_i^{q_i})=\prod f(p_i^{q_i})$
- 常见的积性函数：常函数 $1(n)$、单位函数 $id(n)$、欧拉函数 $\varphi(n)$、莫比乌斯函数 $\mu(n)$、约数函数 $\sigma_x(n)$

如何使用筛法求积性函数

- 求积性函数 f 的关键在于如何快速求 $f(p^k)$
- euler 筛过程中，在筛掉 n 的时候我们也得到了 n 的最小质因数 p
- 我们希望知道 p 在 n 中的次数 k，这样就能利用 $f(n)=f(p^k)f\left(\frac{n}{p^k}\right)$ 求出 $f(n)$
- 令 $n=pm$，那么如果 $p\nmid m$，则 p 在 n 的次数为 1
- 若 $p\mid m$，那么 p 也是 m 的最小质因数，p 在 n 的次数为 p 在 m 的次数 $+1$
- 在筛法时记录每个数的最小质因数的次数，就能算出新筛去合数的最小质因数次数。
- 若 $f(p^k)$ 可以在 $O(1)$ 时间算得，我们可以用 $O(n)$ 的线性时间求一个积性函数的点值

另一道水题[2]

- 给定整数 N，求 $1 \le x, y \le N$ 且 $Gcd(x,y)$ 为素数的数对 (x,y) 有多少对.
- $N \le 10^7$
- 欢迎秒题

[1]https://www.lydsy.com/JudgeOnline/problem.php?id=2818

解答

- 统计 gcd 为素数的对, 考虑枚举素数 p，计算 gcd 恰好为 p 的数对个数
- 即 $\sum_p \sum_{i=1}^{N} \sum_{j=1}^{N} [gcd(i,j) = p]$
- 实际上就是 $\sum_p \sum_{i=1}^{\lfloor N/p \rfloor} \sum_{j=1}^{\lfloor N/p \rfloor} [gcd(i,j) = 1]$
- 统计一定范围内互质数对个数?
- 欧拉函数定义：$\varphi(n)$ 为 $[1,n]$ 内与 n 互质数字个数，即 $\varphi(j) = \sum_{i=1}^{j} [gcd(i,j) = 1]$
- 那么显然 $\sum_{i=1}^{\lfloor N/p \rfloor} \sum_{j=1}^{\lfloor N/p \rfloor} [gcd(i,j) = 1] = -1 + 2\sum_{i=1}^{\lfloor N/p \rfloor} \varphi(i)$(减 1 是因为 (1,1) 被统计了两次)
- 欧拉函数是积性函数，用 euler 筛预处理出点值，求前缀和。枚举质数求和即可在 $O(n)$ 时间解决问题。

上古魔法第一课：杜教筛

- 设 $f(n)$ 是一个数论函数，需要计算 $S(n) = \sum_{i=1}^{n} f(i)$
- n 可能很大，例如 $n \leq 10^{11}$
- 显然求出所有 $f(i)$ 再求和的 $O(n)$ 做法行不通了

一个简单的例子

$$\sum_{i=1}^{n} \sigma_1(i), n \leq 10^{12}$$

- 其中

$$\sigma_1(n) = \sum_{d|n} d = \sum_{i=1}^{n} [i|n] \cdot i$$

- 即求前 n 个正整数的约数之和，例如 6 的约数有 $1,2,3,6$，故 $\sigma_1(6) = 12$

一个简单的例子

$$\sum_{i=1}^{n} \sigma_1(i), n \le 10^{12}$$

- 其中

$$\sigma_1(n) = \sum_{d|n} d = \sum_{i=1}^{n} [i|n] \cdot i$$

- 即求前 n 个正整数的约数之和，例如 6 的约数有 $1,2,3,6$，故 $\sigma_1(6) = 12$
- 推导一下发现

$$\sum_{i=1}^{n} \sigma_1(i) = \sum_{i=1}^{n} \sum_{j=1}^{n} [j|i] \cdot j = \sum_{i=1}^{n} i \cdot \sum_{j=1}^{n} [i|j] = \sum_{i=1}^{n} i \cdot \lfloor \frac{n}{i} \rfloor$$

- 本质上来说这一步转换实际上是枚举每个数，算出它是多少个数的约数，算贡献。

前置知识

- 这个求和式是可以在 $O(\sqrt{n})$ 的时间内求的

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

- 理由是因为 $\lfloor \frac{n}{i} \rfloor$ 取值最多只有 $2\sqrt{n}$ 种不同的取值:
- 对于 $1 \le i \le \sqrt{n}$，由于 i 总共就 $\sqrt{n}$ 种取值，所以 $\lfloor \frac{n}{i} \rfloor$ 不可能超过 $\sqrt{n}$ 种
- 对于 $\sqrt{n} < i \le n$ 由于 $n/i < n/\sqrt{n} = \sqrt{n}$, 所以 $\lfloor \frac{n}{i} \rfloor \le \sqrt{n}$，取值不可能超过 $\sqrt{n}$ 种
- 那么求和式就相当于有很多区间 $[l_k, r_k]$，其中 $\forall i \in [l_k, r_k]$，$\lfloor \frac{n}{i} \rfloor$ 取值相同，这一段的值可以用等差数列求和公式 $O(1)$ 求得 (怎么求 $[l_k, r_k]$ 这样的段的起点终点？)
- 总段数不超过 $O(\sqrt{n})$，所以复杂度是 $O(\sqrt{n})$

又一前置知识：狄利克雷卷积

- 线性筛求积性函数只能算一个简单技巧，OI 比赛中一般不会有这么裸的题。
- 实际上~~毒瘤~~出题人都会给一个式子，要求选手用深厚的数学功底简化，推出积性函数相关的式子。
- 这个时候可能就需要莫比乌斯反演与狄利克雷卷积相关知识解题，题目变种多，公式鬼畜，解题愉悦
- 狄利克雷卷积定义：针对两个函数的运算 $*$
- $$f * g(n) = \sum_{d|n} f(d) \cdot g(\frac{n}{d})$$

抽象：杜教筛到底是什么

- 设 $f(n)$ 是一个数论函数，需要计算 $F(n)=\sum_{i=1}^{n} f(i)$
- 若有数论函数 g,h 使得 $f*g=h$
- 令 F,G,H 为 f,g,h 的前缀和
- 我们有

$$
\begin{aligned}
H(x)=\sum_{n\le x} h(n) = \sum_{n\le x}\sum_{d|n} f(d)g(n/d) \\
= \sum_{d=1}^{x}\sum_{n=1}^{\lfloor x/d\rfloor} f(d)g(n) \\
= \sum_{n\le x} f(n)G(\lfloor x/n\rfloor) = \sum_{n\le x} g(n)F(\lfloor x/n\rfloor)
\end{aligned}
\tag{1}
$$

$$g(1)F(n)=H(n)-\sum_{i=2}^{n} g(i)F(\lfloor n/i\rfloor)$$

简单分析

$$g(1)F(n) = H(n) - \sum_{i=2}^{n} g(i)F(\lfloor n/i \rfloor)$$

- 注意到求和式的右边出现了刚刚提到的只有 $O(\sqrt{n})$ 种取值的 $\lfloor n/i \rfloor$
- 如果我们能够快速求出 $H(x), G(x)$ 就可以快速求出 $F(x)$

naive 杜教筛

$F(n) = H(n) - \sum_{i=2}^{n} g(i)F(\lfloor n/i \rfloor)$
不妨假设 $G(n)H(n)$ 可以在 $O(1)$ 的时间内求出

- 一个小结论：
$$\lfloor \lfloor n/i \rfloor / j \rfloor = \lfloor n/(ij) \rfloor$$
- 这个结论说明，假设我们暴力递归求 $F(n)$，用更小的 $F(\lfloor n/i \rfloor)$ 的值来求，更小的 $F(\lfloor n/i \rfloor)$ 又需要 $F(\lfloor \lfloor n/i \rfloor / j \rfloor)$ 来求时……
- 需要求的 $F(x)$ 的值，即参数 x 的种类数，同样不超过 $\lfloor n/i \rfloor$ 种类数
- 记忆化所有求过的结果, 直接递归记忆化搜索，复杂度为 $O(n^{3/4})$。

复杂度证明

网上能找到的复杂度证明大多含糊不清。
有的复杂度证明表示直接递归，算复杂度时「只需要展开一层就可以了，更深层的复杂度都是高阶小量」，我不懂。
这里给出一个稍微清晰一点的证明（类似动态规划复杂度分析）：

- 递归求 $F(k) = H(k) - \sum_{i=2}^{k} g(i)F(\lfloor k/i \rfloor)$ 时，需要的循环次数为 $O(\sqrt{k})$ 次，即单次转移代价。
- 由于我们已经知道递归求 $F(x)$ 过程中，x 的可能取值 $x = \lfloor\lfloor\lfloor n/i \rfloor / j \rfloor / \cdots / k \rfloor$ 必然是 $\lfloor n/i \rfloor$ 的某一个
- $\lfloor n/i \rfloor$ 的取值总共只有 $O(\sqrt{n})$ 种可能，即状态总数 $O(\sqrt{n})$ 个，且状态空间即 $\lfloor n/i \rfloor$ 的不同取值

那么枚举所有不同取值 x，其转移代价都是 $\sqrt{x}$
于是总复杂度为 $\sum_{i=1}^{\sqrt{n}} \sqrt{i} + \sqrt{n/i}$

复杂度证明（续）

$$\sum_{i=1}^{\sqrt{n}} \sqrt{i} + \sqrt{n/i}$$

显然后半部分比前半部分大，所以只需要估计

$$\sum_{i=1}^{\sqrt{n}} \sqrt{n/i}$$

由一些微积分知识我们有

$$\sum_{i=1}^{\sqrt{n}} \sqrt{n/i} = O\left(\int_0^{\sqrt{n}} \sqrt{n/i}\right) = O\left(n^{3/4}\right)$$

对杜教筛的简单优化

- 打表
- 注意到实际上由于线性筛的存在，我们可以先预处理出一段 $f(n)$ 在 $[1,z]$ 前缀和
- 在递归到求 $x \leq z$ 的 $F(x)$ 时，就不再递归，直接 $O(1)$ 返回对应值。
- 预处理复杂度 $O(z)$
- 我们仅需要对 $x > z$ 的 x 递归算 $F(x)$，这样的 x 是由 $i \leq n/z$ 这样的 $\lfloor n/i \rfloor$ 产生的。
- 所以递归的复杂度，参考之前的复杂度分析，为 $O(\sum_{i=1}^{n/z} \sqrt{n/i})$
- 总复杂度 $O(z + \sum_{i=1}^{n/z} \sqrt{n/i})$
- $\sum_{i=1}^{n/z} \sqrt{n/i} = O(\frac{n}{\sqrt{z}})$，令 $z = n/\sqrt{z} = n^{2/3}$，则复杂度为 $O(n^{2/3})$

简单练习

求

$$\sum_{i=1}^{n} \varphi(i), n \leq 10^{11}$$

- Hint: $\sum_{d|n} \varphi(d) = n$
- 杜教筛要找到函数 g,h， $f * g = h$
- 从而用 $F(n) = H(n) - \sum_{i=2}^{n} g(i)F(\lfloor n/i \rfloor)$ 递归求 $F(n)$
- 能否根据提示给出 g,h 与 F 的递归式?

简单练习

求

$$\sum_{i=1}^{n} \varphi(i), n \leq 10^{11}$$

- Hint: $\sum_{d|n} \varphi(d) = n$
- 杜教筛要找到函数 g,h，$f * g = h$
- 从而用 $F(n) = H(n) - \sum_{i=2}^{n} g(i)F(\lfloor n/i \rfloor)$ 递归求 $F(n)$
- 能否根据提示给出 g,h 与 F 的递归式?
- 显然，$g = 1$，$h = id$，即 $g(x) = 1, h(x) = x$
- 那么 $G(x) = x, H(x) = x(x+1)/2$

$$F(n) = \sum_{i=1}^{n} \varphi(i) = \frac{n(n+1)}{2} - \sum_{i=2}^{n} F(\lfloor n/i \rfloor)$$

对杜教筛的简单优化

- 打表
- 注意到实际上由于线性筛的存在，我们可以先预处理出一段 $f(n)$ 在 $[1,z]$ 前缀和
- 在递归到求 $x \le z$ 的 $F(x)$ 时，就不再递归，直接 $O(1)$ 返回对应值。
- 预处理复杂度 $O(z)$
- 我们仅需要对 $x > z$ 的 x 递归算 $F(x)$，这样的 x 是由 $i \le n/z$ 这样的 $\lfloor n/i \rfloor$ 产生的。
- 所以递归的复杂度，参考之前的复杂度分析，为 $O(\sum_{i=1}^{n/z} \sqrt{n/i})$
- 总复杂度 $O(z + \sum_{i=1}^{n/z} \sqrt{n/i})$
- $\sum_{i=1}^{n/z} \sqrt{n/i} = O(\frac{n}{\sqrt{z}})$，令 $z = n/\sqrt{z} = n^{2/3}$，则复杂度为 $O(n^{2/3})$

另一道练习题

- $\sum_{i=1}^{n} \mu(i)$

另一道练习题解

- 注意到 $\sum_{d|n}\mu(d) = [n = 1]$
- $g = 1, h = e$ 即为所求的 h, g 函数

又一道练习题

- $\sum_{i=1}^{n} \mu(i) i^2$

又一道练习题

- 令 $g(i) = i^2$
- 则 $g * f(n) = \sum_{d|n} \mu(d) d^2 \cdot \left(\frac{n}{d}\right)^2 = \sum_{d|n} \mu(d) = [n = 1]$

杜教筛的局限性

- 显然杜教筛最大的限制就是要找到函数 g,h 使得 $f*g=h$
- 这样的函数并不好找
- 需要一定的经验和数学直觉和~~大量草稿纸~~
- 针对这一问题，扩展埃氏筛可以给出一个较优的解决方案

扩展 Eratosthenes 筛

- 由任之洲同学于 2016 年集训队论文首次引入 OI 界
- 可以用于解决较一般情况下的积性函数求和问题。
- 函数 $f(x)$ 为积性函数。且 $f(p^k)$ 为关于 p,k 的多项式
- 复杂度为 $O\left(\frac{n^{3/4}}{\log n}\right)$
- 俗称“洲阁筛”，本质上为 Eratosthenes 筛的一点拓展

从一个例子看起

- $\phi(n,d) = \prod_{i=1}^{k}(p_i^{c_i} + d)$
- $\phi(1,d) = 1$
- 对给定 n,d 求
$$\sum_{i=1}^{n} \phi(i,d)$$
- 我们记 $\phi(p) = G(p) = p + d, \phi(p^c) = T(p^c) = p^c + d$

简单推导

- 注意到对于 $x \le n$，x 最多只能拥有一个大于 $\sqrt{n}$ 的质因子
- 那么考虑将 $\sum_{i=1}^{n} \phi(i)$ 拆成两部分计算：有大因子的，没有大因子的。

$$\sum_{i=1}^{n} \phi(i) = \sum_{\substack{x \le n, \\ x\text{没有大于}\sqrt{n}\text{素因子}}} \phi(x) \left(1 + \sum_{\substack{\sqrt{n} < p \le \lfloor \frac{n}{x} \rfloor \\ p\text{为质数}}} G(p)\right)$$

- 注意到括号内的取值只与 $\lfloor \frac{n}{x} \rfloor$ 取值有关
- 故可以按照 $\lfloor \frac{n}{x} \rfloor$ 取值将 $\phi(x)$ 分段

设 $y = \lfloor \frac{n}{x} \rfloor$, 即对每段分别计算

$$\sum_{\substack{\sqrt{n} < p \le y \\ p\text{为质数}}} G(p)\text{与} \sum_{\substack{\lfloor \frac{n}{x} \rfloor = y \\ x\text{没有大于}\sqrt{n}\text{素因子}}} \phi(x)$$

第一部分

$$\sum_{\substack{\sqrt{n} < p \le y \\ p\text{为质数}}} G(p)$$

- $G(p) = p + c$
- 设不超过 $\sqrt{n}$ 的素数有 m 个，依次为 $p_1, \cdots, p_m$
- 设 $g[i][j]$ 为 $[1, j]$ 与前 i 个素数互质的所有数之和，$g[0][j] = j(j+1)/2$
- $[1, j]$ 里与前 $i-1$ 个素数互质且为 p_i 倍数的数的和为 $p_i g[i-1][\lfloor \frac{j}{p_i} \rfloor]$
- 所以 $g[i][j] = g[i-1][j] - p_i g[i-1][\lfloor \frac{j}{p_i} \rfloor]$
- $g[m][j] - 1$ 即为 $[1, j]$ 范围内大于 $\sqrt{n}$ 的质数之和。
- 用类似的方法可以算出 $[1, j]$ 范围内大于 $\sqrt{n}$ 的质数的 k 次幂之和
- 注意到我们需要计算的 j 均为 $\lfloor \frac{n}{i} \rfloor$，共有 $\sqrt{n}$ 个

第一部分的暴力

$$g[i][j] = g[i-1][j] - p_i g[i-1][\lfloor \frac{j}{p_i} \rfloor]$$

- 一个结论：$[1,n]$ 内的素数个数为 $O(n/\log n)$ 级别
- 单次状态转移的复杂度为 $O(1)$
- 考虑有效状态数即可算出复杂度
- 注意到第二维度一定是 $\lfloor n/x \rfloor$ 的形式
- 其中对于 $\lfloor n/x \rfloor > \sqrt{n}$ 的第二维度，第一维度需要转移 $\sqrt{n}$ 内的所有质数
- 对于 $\lfloor n/x \rfloor \leq \sqrt{n}$，第一维度只需要转移 n/x 内的所有质数
- 所以总复杂度为

$$\sum_{i=1}^{\sqrt{n}} O\left(\frac{i}{\log i}\right) + \sum_{i=1}^{\sqrt{n}} O\left(\frac{\sqrt{n}}{\log \sqrt{n}}\right) \approx O\left(\frac{n}{\log n}\right)$$

第一部分的优化

$$g[i][j] = g[i-1][j] - p_i g[i-1][\lfloor \frac{j}{p_i} \rfloor]$$

- 朴素 $O(n/\log n)$ 太慢
- 进行一些有理有据的「常数优化」
- 若 $p_{i+1} > j$，那么 $g[i][j] = 1$

第一部分的优化

$$g[i][j] = g[i-1][j] - p_i g[i-1][\lfloor \frac{j}{p_i} \rfloor]$$

- 朴素 $O(n/\log n)$ 太慢
- 进行一些有理有据的「常数优化」
- 若 $p_{i+1} > j$，那么 $g[i][j] = 1$
- 若 $p_i^2 > j \geq p_i$，则 $k = \lfloor \frac{j}{p_i} \rfloor < p_i, g[i-1][k] = 1$。所以 $g[i][j] = g[i-1][j] - p_i$

第一部分的优化

$$g[i][j] = g[i-1][j] - p_i g[i-1][\lfloor \frac{j}{p_i} \rfloor]$$

- 朴素 $O(n/\log n)$ 太慢
- 进行一些有理有据的「常数优化」
- 若 $p_{i+1} > j$，那么 $g[i][j] = 1$
- 若 $p_i^2 > j \geq p_i$，则 $k = \lfloor \frac{j}{p_i} \rfloor < p_i, g[i-1][k] = 1$。所以 $g[i][j] = g[i-1][j] - p_i$
- 计算时可以不用计算 $p_i^2 > j$ 的 i，并记录对于每个 j 最后一次有效转移时的 i，在计算别的 $g[i'][j']$ 用到 $g[i][j]$ 时一并计算 (减去对应的素数之和)
- 那么对于 $\lfloor n/x \rfloor$ 只需要转移不超过 $\lfloor n/x \rfloor$ 的素数

优化后的第一部分复杂度

$$g[i][j] = g[i-1][j] - p_i g[i-1][\lfloor \frac{j}{p_i} \rfloor]$$

- 那么对于 $\lfloor n/x \rfloor$ 只转移不超过 $\lfloor n/x \rfloor$ 的素数
- 总复杂度为

$$\sum_{i=1}^{\sqrt{n}} O\left(\frac{\sqrt{i}}{\log \sqrt{i}}\right) + \sum_{i=1}^{\sqrt{n}} O\left(\frac{\sqrt{n/i}}{\log \sqrt{n/i}}\right) \approx O\left(\frac{n^{3/4}}{\log n}\right)$$

第二部分

$$\sum_{\substack{\lfloor \frac{n}{x} \rfloor = y \\ x\text{没有大于}\sqrt{n}\text{的素因子}}} \phi(x)$$

- 相当于求只有前 m 种素因子的满足 $\lfloor \frac{n}{x} \rfloor = y$ 的 $\phi(x)$ 的和 $f[m][y]$
- 设 $f[i][j]$ 为只包含前 i 种素因子，且 $\lfloor \frac{n}{x} \rfloor = j$ 的 $\phi(x)$ 之和
- 注意到 j 的取值同样只有 $O(\sqrt{n})$ 种。
- 考虑由 $f[i-1][j]$ 用质数 p_i 转移，枚举转移幂次 c，设 $l = \lfloor \frac{j}{p_i^c} \rfloor$，对 f[i][l] 的贡献为 $\phi(p_i^c)f[i-1][j]$

第二部分的暴力

$$\sum_{\substack{\lfloor \frac{n}{x} \rfloor = y \\ x\text{没有大于}\sqrt{n}\text{的素因子}}} \phi(x)$$

- 同样计算状态转移代价与状态数，考虑每个 $\lfloor \frac{n}{x} \rfloor$ 有多少质数用来转移
- 枚举 c 对于每个 $\lfloor \frac{n}{x} \rfloor$ 的可能取值数。
- $h(n) = \sum_{k=2}^{\lfloor \log_2 n \rfloor} O\left(n^{1/k}\right) \approx O(\sqrt{n})$
- 则复杂度为

$$\sum_{i=1}^{\sqrt{n}} O\left(\frac{i+h(i)}{\log i}\right) + \sum_{i=1}^{\sqrt{n}} O\left(\frac{\sqrt{n}+h(n/i)}{\log \sqrt{n}}\right) \approx O\left(\frac{n}{\log n}\right)$$

- 依旧需要一些优化

第二部分的优化

- 考虑省去 $p_i^2 > j$ 的运算
- 设 $y = \lfloor \frac{n}{x} \rfloor$，若 $y \le \sqrt{n}$，那么这个情况下

$$1 + \sum_{\substack{\sqrt{n} < p \le \lfloor \frac{n}{x} \rfloor \\ p\text{为质数}}} G(p) = 1$$

- 当 $p_i^2 > y$ 时，$\lfloor yp_i < p_i \le \sqrt{n}$，所以 y 至多用一次 p_i 去更新且更新完的结果对答案贡献倍率一定为 1
- 所以对于质数 p_i,，只转移 $y \ge p_i^2$，设 $l = \lfloor \frac{j}{p_i^c} \rfloor$，如果 $l < p_i^2$，那么之后不对 $g[i][l]$ 进行转移，所以在这个时候计算 $g[i][l]$ 对最终答案的贡献，即统计 $[p_{i+1}, l]$ 范围内的 $G(p)$ 之和，维护每个状态最后一次转移时的 p_i，统计被忽略的一段 $G(p)$ 之和
- 用类似的复杂度分析技巧可以估计出复杂度为 $O\left(\frac{n^{3/4}}{\log n}\right)$

第二部分优化后的复杂度分析

- 枚举 c 对于每个 $\lfloor \frac{n}{x} \rfloor$ 的可能 (p,c) 取值数（即转移分支个数)。
- $h(n) = \sum_{k=2}^{\lfloor \log_2 n \rfloor} O\left(n^{1/k}\right) \approx O(\sqrt{n})$ 表示 $c \geq 2$ 时的转移分支个数
- 由于我们省去了 $p_i^2 > j$ 的考虑，那么就省去了第一层 $c = 1$ 过全部 j 以内素数的问题。
- 则复杂度从

$$\sum_{i=1}^{\sqrt{n}} O\left(\frac{i+h(i)}{\log i}\right) + \sum_{i=1}^{\sqrt{n}} O\left(\frac{\sqrt{n}+h(n/i)}{\log \sqrt{n}}\right) \approx O\left(\frac{n}{\log n}\right)$$

- 变为

$$\sum_{i=1}^{\sqrt{n}} O\left(\frac{h(i)}{\log i}\right) + \sum_{i=1}^{\sqrt{n}} O\left(\frac{h(n/i)}{\log \sqrt{n}}\right) \approx O\left(\frac{n^{3/4}}{\log n}\right)$$

洲阁筛总结

- 对求和函数形态要求较低，可以在 $O\left(\frac{n^{3/4}}{\log n}\right)$ 解决大量积性函数求和问题。
- 有一点点难写，推导记忆有一定难度。
- 常数大
- ~~我知道你们都早就会了~~

yet another 扩展 Eratosthenes 筛

- 由 Min_25(山之内宏彰) 提出
- 好想好写~~不好证~~
- 复杂度，不严谨地，可以认为是 $O\left(\frac{n^{3/4}}{\log n}\right)$，但是常数小很多。
- 可能的别称：Min_25 筛
- 今天讲课的动机，之所以讲筛法也就是为了科普这个

简单转换

- 依旧是对一个积性函数 f 求前缀和 $S=\sum_{i=1}^{n} f(i)$
- 积性函数在 p^c 的取值是一个多项式
- 令 $minprime_i$ 表示能整除 i 的最小质数，即 i 的最小质因数。
- 于是

$$\sum_{i=1}^{n} f(i) = 1 + \sum_{\substack{2 \le p^c \le n, \\ p\text{是质数}}} f(p^c)\left(1 + \sum_{\substack{minprime_x > p \\ 2 \le x \le \lfloor \frac{n}{p^c} \rfloor}} f(x)\right)$$

- 即提出最小质因子加速计算。

简单转换

- 注意到显然对于合数 i 有 $minprime_i \leq \sqrt{n}$,
- 所以可以拆成

$$\sum_{\substack{2 \leq p^c \leq n,\\ p\text{是质数}\\ p \leq \sqrt{n}}} f(p^c)\left(1 + \sum_{\substack{minprime_x > p\\ 2 \leq x \leq \lfloor \frac{n}{p^c} \rfloor}} f(x)\right) + \sum_{\substack{p\text{是质数}\\ \sqrt{n} < p \leq n}} f(p)$$

$$g_{n,m} = \sum_{\substack{minprime_x > m\\ 2 \leq x \leq n}} f(x)$$

$$h_n = \sum_{\substack{p\text{是质数}\\ 2 < p \leq n}} f(p)$$

- 能快速求 $g_{n,m}, h_n$ 就能快速求 $\sum f_i$

简单递归

$$
\begin{aligned}
g_{n,m} &= \sum_{\substack{minprime_x > m \\ 2 \le x \le n}} f(x) \\
&= \sum_{\substack{p^c \le n, \\ p\text{是质数} \\ m < p \le \sqrt{n}}} f(p^c) \left(1 + \sum_{\substack{minprime_x > p \\ 2 \le x \le \lfloor \frac{n}{p^c} \rfloor}} f(x) \right) + \sum_{\substack{p\text{是质数} \\ \sqrt{n} < p \le n}} f(p) \\
&= \sum_{\substack{p^c \le n, \\ p\text{是质数} \\ m < p \le \sqrt{n}}} f(p^c) \left(1 + g_{\lfloor \frac{n}{p^c} \rfloor, p} \right) + h_n - h_{\sqrt{n}}
\end{aligned}
\tag{2}
$$

$g_{n,0}$ 即为所求解。

做法

假设我们已经对所有 i 求出了 h_i，那么直接按照式子递归暴力求 $g_{n,0}$ 即可。

注意到有用的 h_i 一定满足存在正整数 m 使得 $\lfloor \frac{n}{m} \rfloor = i$。取值共有 $O(\sqrt{n})$ 个。假设 $f(p)$ 为一个低次多项式，那么可以对不同次数分别算其对 h_n 的贡献。

故只需考虑

$$h_i = \sum_{\substack{p\text{是质数} \\ 2 < p \le i}} p^k$$

我们可以定义

$$h'_{ij} = \sum_{\substack{p\text{是质数或}minprime_p > p_j \\ 2 < p \le i}} p^k$$

参考筛法，有

$$h'_{ij} = h'_{i,j-1} - p_j^k \left(h'_{\lfloor i/p_j \rfloor, j-1} - h'_{p_j - 1, j-1} \right)$$

做法解释

筛出所有需要求解的 h，然后利用递归式求解，就得到了一个好写代码常数小的积性函数求和代码。
对筛法的解释可以参考一个例子
定义 $h_i=\sum_{p\le i}p^2$，则 $h_2=2^2,h_3=2^2+3^2,h_6=2^2+3^2+5^2$
算法过程，一开始实际上是
$h_2=2^2,h_3=2^2+3^2,h_6=2^2+3^2+4^2+5^2+6^2$
第一步将 2 的平方以上倍数筛去，
$h_6=h_6-2^2(h_3-h_1)=h_6-2^2(2^2+3^2)=h_6-(4^2+6^2)$，即删去当前求和序列中，最小质因数为 2 的和式。
第二步将 3 的平方以上倍数筛去，由于 $6<3^2$，未达成筛去条件 (我们是从 p^2 开始筛的)。
所以最终有 $h_6=2^2+3^2+5^2$

复杂度分析

- 筛法部分的复杂度与洲阁筛非常相似（同样的 $p^2 > n$ 就跳过操作）
- 但是自然很多
- 容易证明复杂度也是

$$\sum_{i=1}^{\sqrt{n}} O\left(\frac{\sqrt{i}}{\log\sqrt{i}}\right) + \sum_{i=1}^{\sqrt{n}} O\left(\frac{\sqrt{n/i}}{\log\sqrt{n/i}}\right) \approx O\left(\frac{n^{3/4}}{\log n}\right)$$

- 暴力递归求 $g_{n,0}$ 这部分的时间复杂度较难分析，总体的复杂度当前只能证明其复杂度可能是近线性的 $\Theta(n^{1-\varepsilon})$，但实际表现非常优秀，我们可以证明在 $n \le 10^{13}$ 的时候，运算次数不超过 $O\left(\frac{n^{3}/4}{\log n}\right)$
- 该部分复杂度证明略

筛法部分正确性证明

- 具体来说，筛法部分实际上是：
- 对于求 h_i ，采用欧拉筛法的思想，初始时将每个数都视为质数，将 $f_j, 2 \le j \le i$ 全部加到 h_i 中。然后从小到大遍历每一质数 p ，令 i 从 n 开始，从大到小执行以下操作：
- $$h_i -= (h_{\lfloor \frac{i}{p} \rfloor} - h_{p-1}) f(p)$$

  直到 $i < p^2$
- 我们注意到每执行完一个质数 p，对于每个 i，h_i 扣除且仅扣除了所有满足 x 为合数且 $minprime_x \le p$ 的 $f(x)$。这一点可以用循环不变式归纳证明。

DIVCNT3[3]

约数个数 $\sigma_0(n)$ 函数为 n 的正约数个数

$$S_3(n) = \sum_{i=1}^{n} \sigma_0(i^3)$$

.

给定 $N \leq 10^{1}1$，10000 组询问，求 $S_3(n)$

[1] http://www.spoj.com/problems/DIVCNT3/

DIVCNT3 解法

- 显然 $f(x) = \sigma_0(x^3)$，那么 $f(p^k) = \sigma_0(p^{3k}) = 3k + 1$ 为低次多项式
- 直接套用 Min_25 筛即可

【UR #13】Sanrd[1]

$f(n)=n$的次大质因子
这里的次大为可重集的次大，即 $40=2*2*2*5$，则
$f(40)=2$
特别的 $f(p)=f(1)=0$
给定 $N \leq 10^11$，求 $\sum_{i=1}^{n} f(i)$

[1] http://uoj.ac/problem/188

Sanrd 解法

- 这里的 f 并不是积性函数, 但不妨碍我们套用 Min_25 筛
- 令 $g_{n,m}$ 为 n 以内，最小质因数大于 p_m 的 $f(i)$ 之和
- 转化为

$$\sum_{\substack{p^c \le n,\\ p\text{是质数}\\ m < p \le \sqrt{n}}} \left([c>1]p + p(\text{[p+1,n/p] 内的质数个数}) + g_{\lfloor \frac{n}{p^c} \rfloor, p}\right)$$

- 直接得解

Min_25 筛的局限性

- 在筛 h_i 时，要求 f_i 为完全积性。
- 例如 μ 就不可以用 Min_25 筛的方法来求
- 初始 $h_4 = \mu(2) + \mu(3) + \mu(4) = -2$
- 筛去 2 后 $h_4 = h_4 - \mu(2)(h_2 - h_1) = h_4 - \mu(2)^2 = -3$ 错误

引用及参考代码

- https://zhuanlan.zhihu.com/p/33544708
- 《一些特殊的数论函数求和问题》朱震霆
- 《积性函数的几种求法》任之洲
- http://www.spoj.com/problems/TEES/
- https://gist.github.com/zimpha/25929b668aed23a8607d233d69