杜教筛

杜教筛被用于处理一类数论函数的前缀和问题。对于数论函数 f，杜教筛可以在低于线性时间的复杂度内计算 $S(n)=\sum_{i=1}^{n} f(i)$。

算法思想

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

对于任意一个数论函数 g，必满足：

$$\begin{aligned}\sum_{i=1}^{n}(f*g)(i) &= \sum_{i=1}^{n}\sum_{d\mid i} g(d)f\left(\frac{i}{d}\right)\\ &= \sum_{i=1}^{n} g(i)S\left(\left\lfloor\frac{n}{i}\right\rfloor\right)\end{aligned}$$

其中 $f*g$ 为数论函数 f 和 g 的 狄利克雷卷积。

略证

$g(d)f\left(\frac{i}{d}\right)$ 就是对所有 $i\le n$ 的做贡献，因此变换枚举顺序，枚举 $d,\frac{i}{d}$（分别对应新的 i,j）

$$\begin{aligned}\sum_{i=1}^{n}\sum_{d\mid i} g(d)f\left(\frac{i}{d}\right) &= \sum_{i=1}^{n}\sum_{j=1}^{\lfloor n/i\rfloor} g(i)f(j)\\ &= \sum_{i=1}^{n} g(i)\sum_{j=1}^{\lfloor n/i\rfloor} f(j)\\ &= \sum_{i=1}^{n} g(i)S\left(\left\lfloor\frac{n}{i}\right\rfloor\right)\end{aligned}$$

那么可以得到递推式：

$$\begin{aligned}g(1)S(n) &= \sum_{i=1}^{n} g(i)S\left(\left\lfloor\frac{n}{i}\right\rfloor\right)-\sum_{i=2}^{n} g(i)S\left(\left\lfloor\frac{n}{i}\right\rfloor\right)\\ &= \sum_{i=1}^{n}(f*g)(i)-\sum_{i=2}^{n} g(i)S\left(\left\lfloor\frac{n}{i}\right\rfloor\right)\end{aligned}$$

假如我们可以构造恰当的数论函数 g 使得：

1. 可以快速计算 $\sum_{i=1}^{n}(f*g)(i)$；
2. 可以快速计算 g 的前缀和，以用数论分块求解 $\sum_{i=2}^{n} g(i)S\left(\left\lfloor\frac{n}{i}\right\rfloor\right)$。

则我们可以在较短时间内求得 $g(1)S(n)$。

注意

无论数论函数 f 是否为积性函数，只要可以构造出恰当的数论函数 g, 便都可以考虑用杜教筛求 f 的前缀和。

如考虑 $f(n) = \mathrm{i}\varphi(n)$, 显然 f 不是积性函数，但可取 $g(n) = 1$, 从而：

$$\sum_{k=1}^{n}(f*g)(k) = \mathrm{i}\frac{n(n+1)}{2}$$

计算 $\sum_{k\le m}(f*g)(k)$ 和 $\sum_{k\le m} g(k)$ 的时间复杂度均为 $O(1)$, 故可以考虑使用杜教筛。

时间复杂度

令 $R(n) = \left\{\left\lfloor\frac{n}{k}\right\rfloor : k = 2,3,\dots,n\right\}$, 我们有如下引理：

引理

对任意的 $m \in R(n)$，我们有 $R(m) \subseteq R(n)$.

证明

令 $m = \left\lfloor\frac{n}{x}\right\rfloor$, 任取 $\left\lfloor\frac{m}{y}\right\rfloor \in R(m)$, 由整除分块/数论分块的 引理 1 可知：

$$\left\lfloor\frac{m}{y}\right\rfloor = \left\lfloor\frac{n}{xy}\right\rfloor \in R(n).$$

设计算 $\sum_{i=1}^{n}(f*g)(i)$ 和 $\sum_{i=1}^{n} g(i)$ 的时间复杂度均为 $O(1)$. 由引理可知：使用记忆化之后，每个 $S(k)$ $(k \in R(n))$ 均只会计算一次。

由整除分块/数论分块的 引理 2 可知 $|R(n)| = 2\sqrt{n} + \Theta(1)$. 设计算 $S(n)$ 的时间复杂度为 $T(n)$, 则：

$$\begin{aligned}
T(n) &= \sum_{k\in R(n)} T(k) \\
&= \Theta(\sqrt{n}) + \sum_{k=1}^{\lfloor\sqrt{n}\rfloor} O(\sqrt{k}) + \sum_{k=2}^{\lfloor\sqrt{n}\rfloor} O\left(\sqrt{\frac{n}{k}}\right) \\
&= O\left(\int_0^{\sqrt{n}}\left(\sqrt{x} + \sqrt{\frac{n}{x}}\right)\mathrm{d}x\right) \\
&= O\left(n^{3/4}\right).
\end{aligned}$$

若我们可以预处理出一部分 $S(k)$, 其中 $k = 1,2,\dots,m$，$m \ge \lfloor\sqrt{n}\rfloor$. 设预处理的时间复杂度为 $T_0(m)$, 则此时的 $T(n)$ 为：

$$
\begin{aligned}
T(n) &= T_0(m) + \sum_{k \in R(n); k > m} T(k) \\
&= T_0(m) + \sum_{k=1}^{\lfloor n/m \rfloor} O\left(\sqrt{\frac{n}{k}}\right) \\
&= O\left(T_0(m) + \int_0^{n/m} \sqrt{\frac{n}{x}} \mathrm{d}x\right) \\
&= O\left(T_0(m) + \frac{n}{\sqrt{m}}\right).
\end{aligned}
$$

若 $T_0(m) = O(m)$（如线性筛），由均值不等式可知：当 $m = \Theta\left(n^{2/3}\right)$ 时，$T(n)$ 取得最小值 $O\left(n^{2/3}\right)$.

伪证一例

设计算 $S(n)$ 的复杂度为 $T(n)$, 则有:

$$T(n)=\Theta\left(\sqrt{n}\right)+O\left(\sum_{i=2}^{\lfloor\sqrt{n}\rfloor}T\left(\left\lfloor\frac{n}{i}\right\rfloor\right)\right)$$

$$\begin{aligned}T\left(\left\lfloor\frac{n}{i}\right\rfloor\right)&=\Theta\left(\sqrt{\frac{n}{i}}\right)+O\left(\sum_{j=2}^{\lfloor\sqrt{n/i}\rfloor}T\left(\left\lfloor\frac{n}{ij}\right\rfloor\right)\right)\\&=O\left(\sqrt{\frac{n}{i}}\right)\end{aligned}$$

Note

$O\left(\sum_{j=2}^{\lfloor\sqrt{n/i}\rfloor}T\left(\left\lfloor\frac{n}{ij}\right\rfloor\right)\right)$ 视作高阶无穷小，从而可以舍去。

故:

$$\begin{aligned}T(n)&=\Theta\left(\sqrt{n}\right)+O\left(\sum_{i=2}^{\lfloor\sqrt{n}\rfloor}\sqrt{\frac{n}{i}}\right)\\&=O\left(\sum_{i=1}^{\lfloor\sqrt{n}\rfloor}\sqrt{\frac{n}{i}}\right)\\&=O\left(\int_0^{\sqrt{n}}\sqrt{\frac{n}{x}}\mathrm{d}x\right)\\&=O\left(n^{3/4}\right)\end{aligned}$$

Bug

问题在于「视作高阶无穷小，从而可以舍去」这一处。我们将 $T\left(\left\lfloor\frac{n}{i}\right\rfloor\right)$ 代入 $T(n)$ 的式子里，有：

$$\begin{aligned}T(n) &= \Theta\left(\sqrt{n}\right) + O\left(\sum_{i=2}^{\lfloor\sqrt{n}\rfloor}\sqrt{\frac{n}{i}}\right) + O\left(\sum_{i=2}^{\lfloor\sqrt{n}\rfloor}\sum_{j=2}^{\lfloor\sqrt{n/i}\rfloor} T\left(\left\lfloor\frac{n}{ij}\right\rfloor\right)\right)\\ &= O\left(\sqrt{n} + \int_0^{\sqrt{n}}\sqrt{\frac{n}{x}}\mathrm{d}x\right) + O\left(\sum_{i=2}^{\lfloor\sqrt{n}\rfloor}\sum_{j=2}^{\lfloor\sqrt{n/i}\rfloor} T\left(\left\lfloor\frac{n}{ij}\right\rfloor\right)\right)\\ &= O\left(n^{3/4}\right) + O\left(\sum_{i=2}^{\lfloor\sqrt{n}\rfloor}\sum_{j=2}^{\lfloor\sqrt{n/i}\rfloor} T\left(\left\lfloor\frac{n}{ij}\right\rfloor\right)\right)\end{aligned}$$

我们考虑 $\sum_{i=2}^{\lfloor\sqrt{n}\rfloor}\sum_{j=2}^{\lfloor\sqrt{n/i}\rfloor} T\left(\left\lfloor\frac{n}{ij}\right\rfloor\right)$ 这部分，不难发现：

$$\begin{aligned}\sum_{i=2}^{\lfloor\sqrt{n}\rfloor}\sum_{j=2}^{\lfloor\sqrt{n/i}\rfloor} T\left(\left\lfloor\frac{n}{ij}\right\rfloor\right) &= \Omega\left(\sum_{i=2}^{\lfloor\sqrt{n}\rfloor} T\left(\left\lfloor\frac{n}{i}\cdot\left\lfloor\sqrt{\frac{n}{i}}\right\rfloor^{-1}\right\rfloor\right)\right)\\ &= \Omega\left(\sum_{i=2}^{\lfloor\sqrt{n}\rfloor} T\left(\left\lfloor\sqrt{\frac{n}{i}}\right\rfloor\right)\right)\end{aligned}$$

由于没有引入记忆化，因此上式中的 $T\left(\left\lfloor\sqrt{\frac{n}{i}}\right\rfloor\right)$ 仍然是 $\Omega\left(\left(\frac{n}{i}\right)^{1/4}\right)$ 的，进而所谓的「高阶无穷小」部分是不可以舍去的。

实际上杜教筛的亚线性时间复杂度是由记忆化保证的。只有使用了记忆化之后才能保证不会出现那个多重求和的项。

例题

问题一

P4213【模板】杜教筛（Sum）

求 $S_1(n)=\sum_{i=1}^{n}\mu(i)$ 和 $S_2(n)=\sum_{i=1}^{n}\varphi(i)$ 的值，$1\le n<2^{31}$.

莫比乌斯函数前缀和

我们知道：

$$\epsilon = [n=1] = \mu * 1 = \sum_{d\mid n}\mu(d)$$

$$
\begin{aligned}
S_1(n) &= \sum_{i=1}^{n}\epsilon(i) - \sum_{i=2}^{n} S_1\left(\left\lfloor\frac{n}{i}\right\rfloor\right) \\
&= 1 - \sum_{i=2}^{n} S_1\left(\left\lfloor\frac{n}{i}\right\rfloor\right)
\end{aligned}
$$

时间复杂度的推导见 时间复杂度 一节。

对于较大的值，需要用 `map` / `unordered_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。

莫比乌斯反演

$$
\begin{aligned}
\sum_{i=1}^{n}\sum_{j=1}^{n}[\gcd(i,j)=1] &= \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{d|i,d|j}\mu(d) \\
&= \sum_{d=1}^{n}\mu(d)\left\lfloor\frac{n}{d}\right\rfloor^2
\end{aligned}
$$

由于题目所求的是 $\sum_{i=1}^{n}\sum_{j=1}^{i}[\gcd(i,j)=1]$, 所以我们排除掉 $i=1,j=1$ 的情况，并将结果除以 2 即可。

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 $O\left(n^{\frac{2}{3}}\right)$.

杜教筛

求 $S(n)=\sum_{i=1}^{n}\varphi(i)$.

同样的，$\varphi * 1 = \mathrm{id}$, 从而：

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

代码实现

```cpp
#include <cstring>
#include <iostream>
#include <map>
using namespace std;
constexpr int MAXN = 2000010;
long long T, n, pri[MAXN], cur, mu[MAXN], sum_mu[MAXN];
bool vis[MAXN];
map<long long, long long> mp_mu;

long long S_mu(long long x) {  // 求mu的前缀和
  if (x < MAXN) return sum_mu[x];
  if (mp_mu[x]) return mp_mu[x];  // 如果map中已有该大小的mu值，则可直接返回
  long long ret = (long long)1;
  for (long long 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;  // 路径压缩，方便下次计算
}

long long S_phi(long long x) {  // 求phi的前缀和
  long long ret = (long long)0;
  long long j;
  for (long long i = 1; i <= x; i = j + 1) {
    j = x / (x / i);
    ret += (S_mu(j) - S_mu(i - 1)) * (x / i) * (x / i);
  }
  return (ret - 1) / 2 + 1;
}

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  cin >> T;
  mu[1] = 1;
  for (int i = 2; i < MAXN; i++) {  // 线性筛预处理mu数组
    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;
      }
    }
```

```
  }
  for (int i = 1; i < MAXN; i++)
    sum_mu[i] = sum_mu[i - 1] + mu[i];  // 求mu数组前缀和
  while (T--) {
    cin >> n;
    cout << S_phi(n) << ' ' << S_mu(n) << '\n';
  }
  return 0;
}
```

问题二

「LuoguP3768」简单的数学题

大意：求

$$\sum_{i=1}^{n}\sum_{j=1}^{n} i\cdot j\cdot \gcd(i,j) \pmod p$$

其中 $n\le 10^{10},5\times 10^8\le p\le 1.1\times 10^9$,$p$ 是质数。

利用 $\varphi * 1 = \mathrm{id}$ 做莫比乌斯反演化为：

$$\sum_{d=1}^{n} F^2\left(\left\lfloor\frac{n}{d}\right\rfloor\right)\cdot d^2\varphi(d)$$

其中 $F(n)=\frac{1}{2}n(n+1)$

对 $\sum_{d=1}^{n} F\left(\left\lfloor\frac{n}{d}\right\rfloor\right)^2$ 做数论分块，$d^2\varphi(d)$ 的前缀和用杜教筛处理：

$$f(n)=n^2\varphi(n)=(\mathrm{id}^2\varphi)(n)$$

$$S(n)=\sum_{i=1}^{n} f(i)=\sum_{i=1}^{n}(\mathrm{id}^2\varphi)(i)$$

需要构造积性函数 g，使得 $f\times g$ 和 g 能快速求和。

单纯的 φ 的前缀和可以用 $\varphi * 1$ 的杜教筛处理，但是这里的 f 多了一个 id^2，那么我们就卷一个 id^2 上去，让它变成常数：

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

化一下卷积：

$$
\begin{aligned}
((\mathrm{id}^2\varphi)*\mathrm{id}^2)(i) &= \sum_{d\mid i}\left(\mathrm{id}^2\varphi\right)(d)\,\mathrm{id}^2\left(\frac{i}{d}\right)\\
&= \sum_{d\mid i} d^2\varphi(d)\left(\frac{i}{d}\right)^2\\
&= \sum_{d\mid i} i^2\varphi(d) = i^2\sum_{d\mid i}\varphi(d)\\
&= i^2(\varphi*1)(i) = i^3
\end{aligned}
$$

再化一下 $S(n)$:

$$
\begin{aligned}
S(n) &= \sum_{i=1}^{n}\left((\mathrm{id}^2\varphi)*\mathrm{id}^2\right)(i) - \sum_{i=2}^{n}\mathrm{id}^2(i)S\left(\left\lfloor\frac{n}{i}\right\rfloor\right)\\
&= \sum_{i=1}^{n} i^3 - \sum_{i=2}^{n} i^2 S\left(\left\lfloor\frac{n}{i}\right\rfloor\right)\\
&= \left(\frac{1}{2}n(n+1)\right)^2 - \sum_{i=2}^{n} i^2 S\left(\left\lfloor\frac{n}{i}\right\rfloor\right)
\end{aligned}
$$

分块求解即可。

代码实现

```cpp
// 不要为了省什么内存把数组开小,会卡80
#include <cmath>
#include <iostream>
#include <map>
using namespace std;
constexpr int N = 5e6, NP = 5e6, SZ = N;
long long n, P, inv2, inv6, s[N];
int phi[N], p[NP], cnt, pn;
bool bp[N];
map<long long, long long> s_map;

long long ksm(long long a, long long m) {  // 求逆元用
  long long res = 1;
  while (m) {
    if (m & 1) res = res * a % P;
    a = a * a % P, m >>= 1;
  }
  return res;
}

void prime_work(int k) {  // 线性筛phi, s
  bp[0] = bp[1] = true, 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]] = true;
      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) {  // 立方和
  return k %= P, (k * (k + 1) / 2) % P * ((k * (k + 1) / 2) % P)
% P;
}

long long s2(long long k) {  // 平方和
  return k %= P, k * (k + 1) % P * (k * 2 + 1) % P * inv6 % P;
}

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() {
  cin.tie(nullptr)->sync_with_stdio(false);
  cin >> 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);
  cout << solve();
  return 0;
}
```

参考资料

1. 任之洲，2016，《积性函数求和的几种方法》，2016 年信息学奥林匹克中国国家队候选队员论文
2. 杜教筛的时空复杂度分析 - riteme.site