JZPKIL解题报告

南京师范大学附属中学　顾昱洲

Contents

1 题目大意 **2**

2 基础知识 **2**

2.1 数论函数 2

2.2 常用数论算法 3

2.2.1 Miller-Rabin算法 3

2.2.2 Pollard's rho算法 3

2.3 组合数及逆元的计算 3

2.3.1 单个逆元计算 4

2.3.2 $1 \dots n$的逆元的计算 4

2.3.3 组合数计算 5

3 化简$f_{x,y}(n)$ **5**

4 计算$h_k(n)$ **6**

4.1 $O(k_m^3)$的算法 6

4.2 $O(k_m^2)$的算法 6

4.3 一个简明优美的方法 7

5 求$(\mathrm{id}^x * (\mu \mathrm{id}^y) * \mathrm{id}^z)(n)$ **8**

6 30%的算法 **8**

7 另30%的算法 **8**

8 100%的算法 **9**

1 题目大意

给定n, x, y, 求$(\sum_{1\le i\le n}(i,n)^x[i,n]^y) \bmod 1000000007$. 其中$(a,b)$为$a$与$b$的最大公约数, $[a,b]$为a与b的最小公倍数. 多组询问。

30%的数据满足$x=y$;

另30%的数据满足$n\le 10^9, x, y\le 100$;

100%的数据满足$n\le 10^{18}, 0\le x, y\le 3000$，询问个数$T\le 100$。

2 基础知识

解决这道题需要一定的数论及组合基础。集训队员应当已经掌握这些知识。为了使得NOI铜牌线的同学能够理解本解题报告，这里简单介绍需要的基础知识。

本部分中的定理大部分是显然的，因此这里不给出证明。以下内容中p表示素数。

2.1 数论函数

定义1. **数论函数**是指定义域为正整数，陪域为复数的函数，即$f:\mathbb{Z}^+\to\mathbb{C}$。

定义2. 常见的数论函数:

常函数c 返回值总为c的常函数*(*这里的c既表示返回的值，又表示函数名*)*;

恒等函数$\mathrm{id}(n)$ $\mathrm{id}(n)=n$;

单位函数$e(n)$ 若$n=1$, $e(n)=1$; 否则$e(n)=0$;

欧拉函数$\phi(n)$ 1到n中与n互质的数的个数;

*Möbius*函数$\mu(n)$ 若n有平方因子，则$\mu(n)=0$; 否则设$n=p_1p_2\dots p_k$，有$\mu(n)=(-1)^k$。

定义3. 一个数论函数f被称为是**完全积性函数**，当且仅当$x,y\in\mathbb{Z}^+\Rightarrow f(xy)=f(x)f(y)$;

一个数论函数f被称为是**积性函数**，当且仅当$x,y\in\mathbb{Z}^+,(x,y)=1\Rightarrow f(xy)=f(x)f(y)$。其中$(x,y)$为$x$与$y$的最大公约数。

定理1. 1, id, e, ϕ, μ 均为积性函数，其中1, id与e是完全积性函数。

定理2. 若f为积性函数，那么对于任意$n\ge 2$，令$n=\prod_{1\le i\le k}{p_i}^{q_i}$，则$f(n)=\prod_{1\le i\le k}f({p_i}^{q_i})$

根据这个定理我们可以方便地计算积性函数的函数值。

定义4. 数论函数f与g的*Dirichlet*积$(f*g)(n)=\sum_{d|n}f(d)g(n/d)$

定理3. *Dirichlet*积满足交换律、结合律，对于加法满足分配律。

定理4. 若f与g均为积性函数，那么:

f与g的点积fg为积性函数;

f与g的*Dirichlet*积$f*g$为积性函数。

定理5. *Möbius*反演: $g(n)=\sum_{d|n}f(d)\Rightarrow f(n)=\mu*g$

2.2 常用数论算法

2.2.1 Miller-Rabin算法

Miller-Rabin算法是由Gary L. Miller [2]提出，Micheal O. Rabin [3]修改的一种快速判断质数的方法。

算法思想是取一个随机数a，设$n-1=2^s d$，其中d为奇数，那么当$a^d \neq 1 \pmod n$且对于任何$0 \leq r \leq s-1$有$a^{2^r d} \neq -1 \pmod n$时，$n$是合数；否则$n$有3/4的概率是质数。重复多次以获得更好的结果。

Algorithm 1 MillerRabin(n, a)

```
d ← n − 1, s ← 0
while d mod 2 == 0 do
  d ← d/2, s ← s + 1
end while
w ← powerMod(a, d, n)
if w = 1 then
  return probablyPrime
end if
for i = 0 → r − 1 do
  if w = n − 1 then
    return probablyPrime
  end if
  w ← w^2 (mod n)
end for
return Composite
```

在实际应用中，a取前9个质数可以保证在10^{18}的范围内不会出错。

2.2.2 Pollard's rho算法

Pollard's rho算法是由John Pollard [1]提出的一种快速分解正整数的方法。

算法思想是随机取多项式f(实践中常用$f(x)=x^2+C$)，计算数列$a_{i+1}=f(a_i)$。

对于n的某个非平凡因子d，若存在$a_i = a_j \pmod d, i \neq j$，那么$(|a_i - a_j|, n)$是$n$的非平凡因子。我们要求最小的一组$i, j$，可以使用Floyd找环算法。由于$d$未知，因此我们每次都要计算$(|a_i - a_j|, n)$。

(这个程序对于$n=4$会失效，只要特判就可以了。)

时间复杂度可以被证明是$O(n^{1/4}\text{polylog}n)$的。

2.3 组合数及逆元的计算

我们这里只介绍本题中可能会用到的计算。因此模数为质数P，且除法中分母模P永远不

Algorithm 2 rho(n)

```
if prime(n) then
  addFactor(n)
  return
end if
while True do
  x ← 2, y ← 2, d ← 1, f ← pseudoRandomPolynomial()
  while d = 1 do
    x ← f(x), y ← f(f(y)), d = (|x − y|, n)
  end while
  if d ≠ n then
    rho(d), rho(n/d)
  end if
end while
```

可能是0。组合数C_n^k的n与k均很小。

2.3.1 单个逆元计算

要求$y = x^{-1}$，则有

$$\begin{aligned} xy &= 1 \pmod P \\ xy &= 1 - kP \\ xy + kP &= 1 \end{aligned}$$

只要解出y就可以了。可以用扩展Euclid算法。

时间复杂度$O(\log P)$。

2.3.2 $1 \ldots n$的逆元的计算

首先，显然1的逆元就是1。假设我们已经知道了$1 \ldots n-1$的逆元，要求n的逆元。假设$P = nt + k$，则有

$$\begin{aligned} nt &= -k \pmod P \\ ntk^{-1} &= -1 \pmod P \\ n^2t^2(k^{-1})^2 &= 1 \pmod P \\ n^{-1} &= nt^2(k^{-1})^2 \pmod P \end{aligned}$$

时间复杂度$O(n)$。

2.3.3 组合数计算

假设我们要求$C_n^k \pmod P$，那么

$$C_n^k = \frac{n!}{k!(n-k)!} = n!k!^{-1}(n-k)!^{-1} \pmod P$$

假设n的可能的最大值为n_0，$n! \bmod P$可以$O(n_0)$预处理得出。考虑到$n!^{-1} = (n+1)(n+1)!^{-1}$，我们可以求出$n_0!^{-1}$，然后倒推得到$1 \dots n_0$的$n!^{-1}$。

时间复杂度$O(n_0) - O(1)$。

3 化简$f_{x,y}(n)$

令所求式为$f_{x,y}(n)$，那么

$$\begin{aligned}
&f_{x,y}(n) \\
&= \sum_{1\le i\le n} (i,n)^x [i,n]^y \\
&= \sum_{1\le i\le n} (in)^y (i,n)^{x-y} \\
&= n^y \sum_{1\le i\le n} i^y (i,n)^{x-y} \\
&= n^y \sum_{d|n} d^x \sum_{1\le i\le \frac{n}{d},(i,\frac{n}{d})=1} i^y \\
&= n^y \sum_{d|n} d^x g_y(\frac{n}{d}) \\
&= n^y (\mathrm{id}^x * g_y)(n) \\
&= n^y (\mathrm{id}^x * ((\mu \mathrm{id}^y) * h_y))(n) \\
&= n^y (\mathrm{id}^x * (\mu \mathrm{id}^y) * h_y)(n)
\end{aligned}$$

其中

$$\begin{aligned}
&g_k(n) \\
&= \sum_{1\le i\le n,(i,n)=1} i^k \\
&= \sum_{1\le i\le n} i^k e((i,n)) \\
&= \sum_{1\le i\le n} i^k \sum_{d|(i,n)} \mu(d) \\
&= \sum_{d|n} \mu(d) d^k \sum_{1\le i\le \frac{n}{d}} i^k \\
&= \sum_{d|n} \mu(d) d^k h_k(\frac{n}{d}) \\
&= ((\mu \mathrm{id}^k) * h_k)(n)
\end{aligned}$$

其中

$$h_k(n) = \sum_{1 \le i \le n} i^k$$

是一个关于n的$k+1$次多项式(这一点在下一节中证明)。

根据Dirichlet积关于加法的分配律，我们知道$f_{x,y}(n)$是不超过$y+2$个积性函数的和。

4 计算$h_k(n)$

$h_k(n)$为关于n的$k+1$次多项式并不是显然的，是需要证明的。并且如何计算它也是一个问题。

4.1 $O(k_m^3)$的算法

标题中的k_m表示k的最大值，本题中即为3000，以下同。

$$\begin{aligned}
&(n+1)^{k+1} \\
&= \sum_{0 \le i \le n} (i+1)^{k+1} - i^{k+1} \\
&= \sum_{0 \le i \le n} \sum_{0 \le j \le k} C_{k+1}^j i^j \\
&= \sum_{0 \le j \le k} C_{k+1}^j \sum_{0 \le i \le n} i^j \\
&= 1 + \sum_{0 \le j \le k} C_{k+1}^j \sum_{1 \le i \le n} i^j \\
&= 1 + \sum_{0 \le j \le k} C_{k+1}^j h_j(n)
\end{aligned}$$

(这个推导假定了$0^0 = 1$)

因此$h_k(n) = \frac{1}{k+1}((n+1)^{k+1} - 1 - \sum_{0 \le j < k} C_{k+1}^j h_j(n))$。通过数学归纳法可以证明之前的遗留问题：$h_k(n)$为关于$n$的$k+1$次多项式。并且根据这个递推式暴力计算可以在$O(k_m^3)$的时间复杂度内得到$1 \le k \le k_m$的所有$h_k(n)$的各项系数。空间复杂度$O(k_m^2)$。

4.2 $O(k_m^2)$的算法

为了描述方便，以下我们省略$h_k(n)$中的(n)，记为h_k。

注意到$h_k = \frac{1}{k+1}((n+1)^{k+1} - 1 - \sum_{0 \le j < k} C_{k+1}^j h_j)$中出现了$n+1$的幂，所以为了方便，我们假设$h_0 = n+1$，而非$n$。

又，将h_k写成关于$n+1$的多项式，我们发现$(n+1)^{k+1}$的系数总是$\frac{1}{k+1}$，因此我们将h_k写成$\sum_{0 \le i \le k} b_{k,i} \frac{(n+1)^{i+1}}{i+1}$的形式。

观察b，可知$\frac{b_{i,j}}{j+1}=\frac{b_{i+1,j+1}}{i+1}$，于是有$b_{i,j}=C_i^j b_{i-j,0}$。

化简h_k，有

$$\begin{aligned}h_k(n)&=n^k+h_k(n-1)\\&=n^k+\sum_{0\le i\le k}\frac{b_{k,i}}{i+1}n^{i+1}\\&=n^k+\sum_{0\le i\le k}\frac{C_k^i b_{k-i,0}}{i+1}n^{i+1}\end{aligned}$$

这样，在求出各$b_{i,0}$后，可以在$O(k)$的时间复杂度内求出h_k的各项系数。

又，因为h_k的常数项必然为0，因此$0=\sum_{0\le i\le k}\frac{1}{i+1}b_{k,i}=\sum_{0\le i\le k}\frac{C_k^i}{i+1}b_{k-i,0}$。由此我们可以得到$b_{k,0}$的递推式。

这样，我们可以在$O(k_m^2)$的时间复杂度内计算出所有$0\dots k_m$的所有$b_{k,0}$，然后对于每次询问，在$O(k)$的时间复杂度内求出h_k的各项系数。空间复杂度是$O(k_m)$。

4.3 一个简明优美的方法

上面的方法中有个结论没有证明，而是一个猜想。下面我们介绍一个简明优美的方法 [4]，得到与上面相同的结果。

我们先设出数列$B_1,B_2,\dots$，并记B_i为B^i。这里的指数不表示i次方，而仅是方便记录的符号。我们定义二项式定理：

$$(B+n)^k=n^k+\sum_{1\le i\le k}C_k^i n^{k-i}B^i$$

同时，有

$$(B+n-1)^k=n^k+\sum_{1\le i\le k}C_k^i n^{k-i}(B-1)^i$$

两式相减，得

$$(B+n)^k-(B+n-1)^k=kn^{k-1}+\sum_{2\le i\le k}C_k^i n^{k-i}(B^i-(B-1)^i)$$

现在我们定义B_i，使得对于任意$i\ge 2$，有$B^i=(B-1)^i$。这样就有

$$(B+n)^k-(B+n-1)^k=kn^k$$

两边对于n求和，于是有

$$\begin{aligned} h_{k-1}(n) &= \sum_{1\le i\le n} n^{k-1} \\ &= \frac{1}{k}\sum_{1\le i\le n} kn^{k-1} \\ &= \frac{1}{k}\sum_{1\le i\le n} ((B+i)^k-(B+i-1)^k) \\ &= \frac{(B+n)^k-B^k}{k} \end{aligned}$$

结果与我们之前的做法符合。

5 求$(\mathrm{id}^x * (\mu\mathrm{id}^y) * \mathrm{id}^z)(n)$

根据Dirichlet积的性质，我们知道$\mathrm{id}^x * (\mu\mathrm{id}^y) * \mathrm{id}^z$是一个积性函数。那么将$n$分解，$n=\prod_{1\le i\le k} p_i^{q_i}$，现在只需求$(\mathrm{id}^x * (\mu\mathrm{id}^y) * \mathrm{id}^z)(p^k)$(这里的$k$和$n$的分解中的$k$没有任何关系)即可。

考虑到μ的自变量指数大于1时值为0，因此我们可以对于p^k分配到$\mu\mathrm{id}^y$项的指数进行讨论，只有0和1两种情况。

现在问题变为求$(\mathrm{id}^x * \mathrm{id}^z)(p^k)$。我们考虑分配到两项中的指数，发现所有情况构成了等比数列。我们要做的是一个等比数列求和。

等比数列求和有三种方法：暴力求和、使用矩阵乘法、利用等比数列求和公式计算。由于本题中k较小，因此暴力求和效果不坏。

6 30%的算法

30%的数据中$x=y$，因此$f_{x,y}(n)=\sum_{1\le i\le n}(i,n)^x[i,n]^x=\sum_{1\le i\le n}(in)^x=n^x h_x(n)$。因此我们只需要求$h_x(n)$的各项系数即可。

时间复杂度$O(k_m^2+Tk_m)$。空间复杂度$O(k_m)$。

7 另30%的算法

另30%的数据中$n\le 10^9, x,y\le 100$，因此我们可以对n暴力分解质因数。而$x,y\le 100$，因此我们可以用$O(k_m^3)$的算法求$h_k(n)$。

时间复杂度$O(T\sqrt{n}/\log n + Tk_m\log n + k_m^3)$。时间复杂度的第一部分是暴力分解质因数(注意不是$T\sqrt{n}$)，第二部分是求$n$因子中所有$p^k$的$\mathrm{id}^x * (\mu\mathrm{id}^y) * \mathrm{id}^z$值，第三部分是预处理$h_k(n)$的各项系数。

空间复杂度$O(k_m^2)$。

8 100%的算法

100%的数据中，暴力分解质因数显然会TLE。我们使用Pollard's rho算法进行分解。并使用$O(k_m^2)$的算法预处理$b_{i,0}$。

时间复杂度$O(Tn^{1/4}\text{polylog}n + Tk_m \log n + k_m^2)$，空间复杂度$O(k_m)$。

参考文献

[1] Pollard, J. M. , A Monte Carlo method for factorization, *BIT Numerical Mathematics* **15** (3): 331 – 334, 1975

[2] Miller, Gary L. , Riemann's Hypothesis and Tests for Primality, *Journal of Computer and System Sciences* **13** (3): 300 – 317, 1976

[3] Rabin, Michael O. , Probabilistic algorithm for testing primality, *Journal of Number Theory* **12** (1): 128 – 138, 1980

[4] Dörrie H. , Bernoulli's Power Sum Problem, *100 Great Problems of Elementary Mathematics: Their History and Solutions*. New York: Dover. pp. 40 – 44 (§ 11), 1965