浅谈 Nimber 和多项式算法

宁波市镇海中学 罗煜翔

摘 要

本文从 Nimber 的定义出发，介绍了 Nimber 的快速运算，Nimber 多项式的乘法、分治算法、复合、牛顿迭代、除法、多点求值、快速插值、欧几里得算法、中国剩余算法、Nimber 二项卷积、ln 和 exp、三角函数这些问题的定义和快速算法。

引言

组合游戏或者说博弈论是 OI 中的一个重要考点，在各大比赛中常常出现。目前的考察重点以 SG 定理为主，但相关的扩展主要以数据结构优化为主。

多项式问题也是 OI 中的一个重要考点，但笔者发现一些多项式方面的算法[1]虽然也已经被引入但并不普及，导致出现了使用其中一些算法的特例出题的情况。

于是，笔者希望通过本文，综合组合游戏和多项式相关的内容，介绍一些相关的算法和问题。

本文的第一节会给出 Nimber 的定义和相关问题，第二节讨论 Nimber 生成函数及其各种运算，第三节讨论 Nimber 指数生成函数及其各种运算。

1 Nimber

1.1 Nimber 和 Sprague-Grundy 定理

1.1.1 组合游戏

首先我们定义一般规则下的公平组合游戏为满足如下条件的组合游戏：

1. 游戏由两个玩家轮流进行，直到无法进行操作时的操作者输。

[1]如有限域上的多项式因式分解

2. 游戏允许进行的操作只取决于当前的状态，不取决于操作者是谁。游戏的所有信息对双方公开。

3. 游戏的总状态和总操作有限，所有的操作对状态的影响是确定性的。

我们可以用后继状态的集合表示游戏中的一个状态。例如无法操作为 $\varnothing$，只能操作到无法操作为 $\{\varnothing\}$。在不引起混淆的情况下，我们也直接用这个状态表示一个组合游戏。

定义单堆的 Nim 游戏为：有一堆石子，每次的操作可以从中取任意正整数数量的石子的一般规则公平组合游戏。

我们定义 Nimber $*n$ 为大小为 n 的石子的单堆 Nim 游戏。形式化的说，Nimber 由以下规则归纳定理 $*0 = \varnothing$ 且对 $n \geq 0$ 有 $*(n+1) = *n \cup \{*n\}$。在不会导致混淆的情况下，$*$ 可以省略。

对于两个组合游戏 A, B，我们可以定义他们的和：

$$A \oplus B = \{a \oplus B | a \in A\} \cup \{A \oplus b | b \in B\}$$

即每次只能在一个游戏中进行操作。显然其满足交换律和结合律。

容易发现，单堆 Nim 游戏的胜负情况只和 n 是否为 0 有关，但不难发现这些状态并不等价。这表明两个组合游戏的等价不能只看胜负情况。

于是我们定义两个组合游戏 A, B 等价，当且仅当对于任意的组合游戏 C，均有 $A \oplus C$ 和 $B \oplus C$ 的胜负情况相同。记作 $A \approx B$。

1.1.2 Sprague-Grundy 定理

Sprague-Grundy 定理指出，任意的公平组合游戏和一个 Nimber 等价。所以公平组合游戏的研究可以转化为 Nimber 的研究。由于和本文内容关系不大，Sprague-Grundy 定理的证明这里省略[2]。

我们将这个组合游戏等价的 Nimber 称为该组合游戏的 SG 函数。计算方法为所有后继状态的 SG 函数中，最小没出现的非负整数[3]。

定义两个 Nimber 的 Nim 和为这两个 Nimber 的游戏的和所等价的 Nimber，不难证明这就是将 Nimber 的数值的按位异或的结果。

1.2 Nim 积

Nim 积，可以看做是 Nim 和的扩展，其定义式如下：

$$x \otimes y = \operatorname{mex}\{(a \otimes b) \oplus (a \otimes y) \oplus (x \otimes b) \mid 0 \leq a < x \wedge 0 \leq b < y\}$$

[2]证明可以参见参考文献 [3]

[3]也称为 mex 运算

这相当于是每次操作把 $(a,b)-(x,y)$ 这个矩形的四个顶点翻转颜色的组合游戏的 SG 函数。

可以证明，所有 $[0,2^{2^m})$ 中的 Nimber 构成了一个有限域，全体的 Nimber 构成代数闭域。证明较为复杂，这里略去。[4]

对于计算两个数的 Nim 积，需要利用如下结论，证明这里同样略去。

$$2^{2^n}\otimes 2^{2^m}=\begin{cases}2^{2^n}\cdot 2^{2^m} & \text{if } n\neq m\\ 3\cdot 2^{2^n-1} & \text{if } n=m\end{cases}$$

考虑利用这一结论求两个数的 Nim 积。

设 $x=a\cdot 2^{2^{m-1}}+b, y=c\cdot 2^{2^{m-1}}+d$，其中 $a,b,c,d\in[0,2^{2^{m-1}}), x,y\in[0,2^{2^m})$。

设 $n=2^{m-1}$，则不难证明有 $2^n\otimes a=2^n\cdot a$，于是：

$$\begin{aligned}x\otimes y=(a\otimes 2^n\oplus b)\otimes(c\otimes 2^n\oplus d)&=a\otimes c\otimes 3\cdot 2^{n-1}\oplus(a\otimes d\oplus b\otimes c)\otimes 2^n\oplus b\otimes d\\&=a\otimes c\otimes 2^{n-1}\oplus(a\otimes d\oplus b\otimes c\oplus a\otimes c)\cdot 2^n\oplus b\otimes d\end{aligned}$$

注意到 $a\otimes d\oplus b\otimes c\oplus a\otimes c=(a\oplus b)\otimes(c\oplus d)\oplus b\otimes d$。

于是进行 4 次 $m-1$ 的递归即可，时间复杂度为 $O(4^m)$。事实上，注意到其中一次递归为 2 的幂和另一个数的 Nim 积。在 x 是 2 的幂的情况下，可以发现上面的 a,b 中有一个是 0，另一个也是 2 的幂。于是进行的 4 次递归中，一次 $x=0$ 从而结果也为 0，另外三次 x 是 2 的幂，则每次只会进行三个递归，时间复杂度为 $O(3^m)$。

于是考虑总的过程，则有 $T(m)=3T(m-1)+O(3^m)$，解得 $T(m)=O(m\cdot 3^m)$。事实上，由于 $O(3^m)$ 实际上是 $m-1$ 的大小的复杂度，所以常数其实只有 $\frac{1}{3}$。

由于这一算法十分的重要，所以这里给出伪代码。事实上，$m\cdot 3^m$ 和 4^m 的差距并没有多大，不过由于代码上只差一个 if，所以这里给出的是 $O(m\cdot 3^m)$ 的版本。

Algorithm 1 NimMul(x,y,m)

if $x=0\vee y=0$ **then**
  return 0
end if
if $m=0$ **then**
  return 1
end if
$n\leftarrow 2^{m-1}, a\leftarrow\left\lfloor\frac{x}{2^n}\right\rfloor, b\leftarrow x \bmod 2^n, c\leftarrow\left\lfloor\frac{y}{2^n}\right\rfloor, d\leftarrow y \bmod 2^n$.
$ac\leftarrow$ NimMul$(a,c,m-1), bd\leftarrow$ NimMul$(b,d,m-1)$.
return $(\text{NimMul}(a\oplus b,c\oplus d,m-1)\oplus bd)\cdot 2^n+\left(\text{NimMul}\left(2^{n-1},ac,m-1\right)\oplus bd\right)$.

[4]相关的证明可见参考文献[2]

注意到 $5 \cdot 3^4 = 405$ 仍然比较大，考虑对这个算法进行进一步的优化。

一个直接的想法是进行记忆化，由于 x, y 共有 $2^{2^{m+1}}$ 种，在 $m \le 3$ 的时候记忆化是可以接受的。

更进一步的想法是注意到有限域中原根[5]的存在，可以预处理 $[0, 2^{2^m})$ 范围内的指数/对数表。于是可以处理 $m \le 4$ 的情况。使用了上述优化后，在 UOJ 上进行 10^7 次 Nim 积大约需要 400ms。

于是 $m = 5$ 的情况只需要一次递归就能得到答案，可以认为时间复杂度是 $O(1)$ 的。由于 $[0, 2^{32})$ 在 C++[6] 中就是 unsigned int 的表示范围，所以下文如无特殊说明只考虑这个范围内的 Nimber。

接下来的部分，如无特别说明，则 a^b 在 a 是具体数字或时间复杂度中时表示数字的乘方，在 a 是变量或者表达式时表示 Nim 幂，即做 b 次 Nim 积后的结果。

1.3 Nimber 开方与求逆

考虑解方程 $x^2 = a$ 和 $a \otimes x = 1$。

由有限域的基本性质，$x^{2^{32}} = x$。

从而一组可行解是 $a^{2^{31}}$ 和 $a^{2^{32}-2}$（$a \neq 0$），利用快速幂计算即可。

解的唯一性是不难证明的。

1.4 Nimber 积和式

给定一个 $n \times n$ 的二维数组 $a_{i,j}$，求 $\bigoplus_p \bigotimes_{i=1}^{n} a_{i,p_i}$，这里的 p 取遍所有 $1 \sim n$ 的排列。

由于 $x \oplus x = 0$，所以积和式就是行列式。

使用高斯消元法计算即可。

1.5 Nimber 多项式方程

现在考虑解一个一般的多项式方程 $\bigoplus_{i=0}^{n} a_i \otimes x^i = 0$。这个方程可能有 $x \ge 2^{32}$ 的根，但这里只考虑求 $[0, 2^{32})$ 中的根。

首先将原多项式和 $x^{2^{32}} \oplus x$ 求 gcd，则求完 gcd 的结果是若干个互不相同的一次式的乘积。

设 $F(x) = \bigoplus_{i=1}^{31} x^{2^i}$，则 $F(x)\,(F(x) \oplus 1) = x^{2^{32}} \oplus x$。

于是随机一个多项式 l，则 $\gcd(F(l(x)), G(x))$ 至少有 $\frac{1}{2}$ 的概率找到 G 的一个非平凡因式。

[5]对于 $m = 4$ 的情况，最小的原根为 258

[6]这里指 NOI Linux 上的 C++

而 $F(l(x))$ 的计算可以类似快速幂的方法计算。

时间复杂度不超过 $O\left(32n^2\log n\right)$。

1.5.1 例1. 小 Q 和小 Y 做游戏（一）

小 Q 和小 Y 是组合带师。

这天，祂们发现了二维平面第一象限中[7]的一个点 (a,b)，于是祂们决定来进行一次游戏。

祂们首先把这个点染成了黑色，并把其他点染成了白色。并从小 Q 开始轮流操作：

选择一个顶点均为非负整数、边平行于坐标轴且右上角是黑点的矩形[8]，将它的四个顶点翻转颜色。如果没有这样的矩形，则操作者输。

由于小 Y 非常希望胜利，所以祂准备作弊。祂可以在小 Q 开始操作前，选择一条第一象限中平行于 x 轴的直线，将它与直线 $x=a$ 与 $y=x$ 的两个交点依次翻转颜色[9]。

于是小 Y 准备寻求你的帮助，希望你求出应该取哪条直线。当然，小 Y 考虑到了你的计算能力，所以如果 $(0,2^{32})$ 中无解则只需要输出无解。

如果有多解求出任意一个均可。

$1\le a,b<2^{32}$。

1.5.2 解法

不难发现，问题相当于解方程 $x^2\oplus a\otimes x\oplus a\otimes b=0$。

普通的解二次方程方法是基于配方的，但对 Nimber 来说配方消不掉一次项。

事实上，这里直接套用 Nimber 多项式方程的解法就行了。

时间复杂度 $O(32)$。

2 Nimber 生成函数

现在，我们考虑 Nimber 多项式和 Nimber 生成函数。

首先要解决的问题就是多项式的乘法。

2.1 Nimber 多项式乘法

主要有三种方法可以解决这个问题。

[7]这里认为坐标轴上的点不在任何象限

[8]这里认为矩形的面积需要为正

[9]如果这两个交点是同一个则什么也不做

2.1.1 Karatsuba 算法

很容易想到可以利用 Karatsuba 算法解决这个问题。

由于 Karatsuba 算法在 OI 中已经比较普及，这里只简单的叙述一下过程。

每次将两个多项式拆成前一半和后一半，全部展开合并同类型后只需要计算最高项，最低项和三项的和，于是进行三次一半大小的递归即可。

时间复杂度为 $O\left(n^{\log_2 3}\right) \approx O\left(n^{1.585}\right)$。

2.1.2 FFT

有限域 $\mathbb{F}_{p^k}$ 的一种构造是：先选择一个模 p 意义下的 k 次既约多项式 $f(x)$，则对于所有模 p 的 p^k 个 $k-1$ 次多项式，乘法定义为模 $p, f(x)$ 的乘法。

在将 Nimber 转换成这种形式后我们可以用 FFT 做二元多项式乘法解决这个问题。

一种转换的方法是，找到 $\mathbb{F}_{2^{32}}$ 的一个原根，设其为 x，利用 $x^0, x^1, \cdots, x^{32}$ 进行高斯消元法解出这个 32 次多项式。

由于二元多项式乘法需要长度扩大 4 倍，且最后还需要做 $O(n)$ 次多项式取模，所以常数比较大，甚至不一定能跑过 Karatsuba 算法。

2.1.3 Cantor 算法

对于一般的多项式乘法问题，有时间复杂度为 $O(n \log n \log\log n)$ 的算法，下面就介绍其中的一种。

简单来说，这个算法直接在 FFT 的过程中维护单位根的和。

由于 $x \oplus x = 0$，这导致无法进行二进制的 IDFT，所以考虑使用三进制。[10]

考虑计算两个多项式乘积模 $x^{2\times 3^m} \oplus x^{3^m} \oplus 1$ 的结果。事实上，取模的多项式是分圆多项式，下面的计算过程的正确性需要用它来保证。

设 $v = \left\lfloor \frac{m}{2} \right\rfloor, u = \left\lceil \frac{m}{2} \right\rceil, n = 3^m, p = 3^u, q = 3^v$，则有：

$$\bigoplus_{i=0}^{2n-1} a_i x^i = \bigoplus_{j=0}^{q-1} \left(\bigoplus_{i=0}^{2p-1} a_{iq+j} x^{iq} \right) \otimes x^j$$

设 $x^q = y$，则可以看做是 x 次数为 $q-1$ 的二元多项式，且 y 的部分需要对 $y^{2p} \oplus y^p \oplus 1$ 取模。

对两个多项式做乘法，由于 $y^{2p} \oplus y^p \oplus 1$ 是 $y^{3p} \oplus 1$ 的因式，可以认为 y 就是 $3p$ 次单位根。

[10]对于一般问题，需要做二、三进制并用扩展欧几里得算法合并

现在有两个长度为 $q-1$ 的多项式，做乘法的直接想法是做长度至少为 $2q-1$ 的循环卷积。由于需要做三进制 FFT，所以需要长度为 $3q$ 的。由于 $p \geq q$，所以 $3q$ 次单位根可以用 y 表示。

考虑做 FFT 需要的操作，交换元素和加减法可以用多项式加减法处理，而乘上单位根只需要做一次平移然后取模。由于取模的多项式非常简单所以都是 $O(p)$ 的。

在做完 DFT 之后，需要对应项做点积，这其实就是 u 大小的子问题，递归处理。

IDFT 和 DFT 是类似的，事实上 IDFT 就是 DFT 后翻转除了首项的一段区间。

时间复杂度 $T(n) = 3\sqrt{n}T(\sqrt{n}) + O(n\log n)$，$T(n) = O\left(n\log^{1+\log_2 3} n\right)$。

发现复杂度不优的原因是明明两边相乘长度只有 $2q$ 却要补到 $3q$，考虑直接做长度为 q 的循环卷积。

设 $C(x) = A(x) \otimes B(x)$，即原来两个多项式的乘积。

考虑计算 $D(x) = A(x) \otimes B(x)$ 和 $E(x) = A(x \otimes \omega_{3q}) \otimes B(x \otimes \omega_{3q})$，这里的卷积是循环卷积。

则 $d_i = c_i \oplus c_{i+q}, e_i \otimes \omega_{3q}^{-i} = c_i \oplus \left(c_{i+q} \otimes \omega_3\right)$。

显然只需要求出 $\omega_3 \oplus 1$ 的逆即可，注意到 $\omega_3 \oplus 1 = \omega_3^2$，所以逆就是 ω_3。

于是，时间复杂度为 $T(n) = 2\sqrt{n}T(\sqrt{n}) + O(n\log n)$，得 $T(n) = O(n\log n \log\log n)$。

使用这个算法进行模素数的多项式乘法，在 UOJ 上对 $n = 100000$ 的数据大约需要 70ms，大约为常数极好的拆系数 FFT/三模数 NTT [11]的 1.5 ~ 2 倍。

2.2 离散 Fourier 变换

虽然 Nimber 数不能做 2 的幂的 DFT，但由于原根存在，依然可以做一些长度的 DFT。

确切地说，对于 $n | 2^{32}-1$，长度为 n 的 DFT 都是可以定义的。而且对于 $0 \leq i \leq 31$，$[2^i, 2^{i+1})$ 中恰好存在一个 n 可以做 DFT。

这一点可以枚举验证，也可以利用 $2^{32}-1$ 的质因子都是费马素数证明。

现在考虑如何求一个一般长度的 DFT：

$$b_i = \bigoplus_{j=0}^{n-1} a_j \otimes \omega_n^{i \cdot j}$$

利用等式 $\binom{i+j}{2} - \binom{i}{2} - \binom{j}{2} = i \cdot j$，得：

$$b_i \otimes \omega_n^{\binom{i}{2}} = \bigoplus_{j=0}^{n-1} a_j \otimes \omega_n^{-\binom{j}{2}} \otimes \omega_n^{\binom{i+j}{2}}$$

于是做一次减法卷积即可，这可以通过翻转 a 变成加法卷积实现。

而 IDFT 也没什么区别，事实上在 DFT 的结果上直接反转 $1 \sim n-1$ 的值即可。

[11]这方面的快速算法可以见参考文献[7]

2.3 Nimber 半在线卷积

定义半在线卷积问题为：

计算 $f = g \otimes h$，其中 g 已经给出且常数项为 0，而 h_i 将在计算出 f_i 后给出[12]。

容易想到用分治算法解决这个问题：

首先递归计算左边一半的 f_i，然后用左边的 h_i 更新右半边的 f_i，最后递归处理右边即可。

时间复杂度为 $O(n \log^2 n \log\log n)$。

考虑每次不是分成两段，而是分成 K 段，则每两段之间都需要进行转移，共有 $O(K^2)$ 次转移。

但是注意到多项式的个数只有 $O(K)$ 种，于是进行 DFT 之后转移就是线性的。

于是每层的时间复杂度就变成了 $O\left(n \log \frac{n}{K} \log\log \frac{n}{K} + nK\right)$，取 $K = O(\log n \log\log n)$ 时，每一层所花的时间还是 $O(n \log n \log\log n)$。但这样每次就可以分为 K 个子问题。

于是总的时间复杂度变为了 $O\left(\frac{n \log^2 n \log\log n}{\log\log n}\right) = O\left(n \log^2 n\right)$。

注意到转移的过程依然是若干次半在线卷积，可以做到 $O\left(n \log^{1+\varepsilon} n\right)$，但由于一些常数原因，实践中和 $O\left(n \log^2 n\right)$ 区别不大。

2.4 Nimber 多项式复合

计算 $(f \circ g)(x) \bmod x^{2^k}$，其中 $f \circ g$ 表示 f 和 g 的复合，即 $f(g)$。

设 $f(x) = f_0\left(x^2\right) \oplus \left(x \otimes f_1\left(x^2\right)\right)$，于是只需要计算 $f_0 \circ g^2, f_1 \circ g^2$。

由于 g^2 的奇数项均为 0，所以可设 $g^2(x) = h\left(x^2\right)$，于是只需要计算 $f_0(h), f_1(h) \bmod x^{2^{k-1}}$。

时间复杂度为 $O\left(n \log^2 n \log\log n\right)$。

2.5 牛顿迭代

给定一个多项式 $F(t)$，你需要求一个多项式 $X(x)$ 使得 $(F \circ X)(x) \bmod x^n = 0$ 且 X 的常数项为 0。要求 X 常数项为 0 才能对形式幂级数比较准确的定义复合，但这可能导致公式中大量出现 $X(x) \oplus 1$。

事实上，传统的牛顿迭代法在这里还是适用的。

首先是牛顿迭代公式 $X_{t+1} = X_t \oplus \frac{F \circ X_t}{F' \circ X_t} \bmod x^{2^{t+1}}$，其中 X_t 表示 $X \bmod x^{2^t}$，F' 为导数，实现上就是去掉偶数项后除以 x。

于是第一个问题做多项式求逆，只需要解方程：$G(x) = \frac{1}{1 \oplus X(x)} = 1 \oplus X(x) \oplus X(x)^2 \oplus \cdots$。

带入上面的公式并化简得：$X_{t+1} = 1 \oplus (X_t \oplus 1)^2 \otimes G \bmod x^{2^{t+1}}$。

[12]可以理解为 h_i 的计算依赖 f_i

时间复杂度为 $O(n\log n\log\log n)$。

于是对于一般的问题，时间复杂度为 $O\left(n\log^2 n\log\log n\right)$。如果函数比较特殊，复合计算较快则为 $O(n\log n\log\log n)$。

2.6 Nimber 多项式带余除法

给定 n 次多项式 $A(x)$ 和 m 次多项式 $B(x)$，要求 $n-m$ 次多项式 $C(x)$ 和次数不超过 $m-1$ 次多项式 $D(x)$，使得

$$A(x)=B(x)\otimes C(x)\oplus D(x)$$

考虑将 x 用 $\frac{1}{x}$ 来代，并在两边乘 x^n，得：

$$x^n\otimes A\left(\frac{1}{x}\right)=\left(x^m\otimes B\left(\frac{1}{x}\right)\right)\otimes\left(x^{n-m}\otimes C\left(\frac{1}{x}\right)\right)\oplus x^{n-m+1}\otimes\left(x^{m-1}\otimes D\left(\frac{1}{x}\right)\right)$$

两遍模 x^{n-m+1}，则只需要进行多项式求逆和乘法就能求出 C，之后求 D 也是容易的。

2.7 Nimber 线性递推

对于递推数列：$f_m=\bigoplus_{i=1}^{d}a_i\otimes f_{m-i}$，给定 $\forall 1\le i\le d,a_i,f_{i-1}$，求 f_n。

考虑构造矩阵：

$$\mathbf{M}=\begin{pmatrix}a_1 & a_2 & \cdots & a_{d-1} & a_d\\ 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \end{pmatrix}$$

我们可以得到 $\mathbf{F}_n=\mathbf{M}^n\otimes\mathbf{F}_0$，其中 $\mathbf{F}_i=(f_{i+d-1},f_{i+d-2},\cdots,f_i)^T$

对一个矩阵 $\mathbf{M}$，我们定义特征多项式 $f_{\mathbf{M}}(\lambda)=\det(\lambda\otimes\mathbf{I}\oplus\mathbf{M})$，则 Cayley-Hamilton 定理表明：$f_{\mathbf{M}}(\mathbf{M})=\mathbf{0}$。

对于这个问题，$f_{\mathbf{M}}(\lambda)=\lambda^d\oplus\bigoplus_{i=1}^{d}a_i\otimes\lambda^{d-i}$，所以 $\mathbf{M}^d\oplus\bigoplus_{i=1}^{d}a_i\otimes\mathbf{M}^{d-i}=\mathbf{0}$。

当 $x^d\oplus\bigoplus_{i=1}^{d}a_i\otimes x^{d-i}=0$ 时，我们有 $x^n=x^n\bmod\left(x^d\oplus\bigoplus_{i=1}^{d}a_i\otimes x^{d-i}\right)=\bigoplus_{i=0}^{d-1}c_i\otimes x^i$。

于是可以用快速幂计算出 $\mathbf{M}^n=\bigoplus_{i=0}^{d-1}c_i\mathbf{M}^i$，则 $\mathbf{F}_n=\mathbf{M}^n\otimes\mathbf{F}_0=\bigoplus_{i=0}^{d-1}c_i\otimes\mathbf{M}^i\otimes\mathbf{F}_0$，从而 $f_n=\bigoplus_{i=0}^{d-1}c_i\otimes f_i$。

2.8 Nimber 多项式欧几里得算法

给定两个 Nimber 多项式，求出它们的最大公因式和 Bézout 系数[13]。

首先考虑下面这个问题：

给定两个次数不超过 $2 \cdot n$ 的多项式，求其辗转相除过程中最后两个度数不小于 n 次的多项式。

首先有如下引理：给定两个次数不超过 n 的多项式，若只考虑次数不小于 k 的部分的辗转相除，则辗转相除的过程[14]是不会受到多项式最低 $2k-n$ 位的影响的。

证明：可以证明 Bézout 系数不超过 $n-k$ 次，于是最后 $2k-n$ 位不会影响到高位的值，从而不影响 Bézout 系数。

于是，先求次数不小于 $1.5n$ 的最后两个，需要递归的长度为 n。然后再求不小于 n 次的最后两个，需要递归的长度还是 n。

总的时间复杂度就是 $T(n)=T\left(\frac{n}{2}\right)+O(n\log n\log\log n)$，即 $T(n)=O\left(n\log^2 n\log\log n\right)$。

在解决了这个问题之后，每次减少一半长度，总时间复杂度为 $O\left(n\log^2 n\log\log n\right)$。

2.9 Nimber 多项式求值

给定 $F(x),x_1,x_2,\cdots,x_n$，计算 $F(x_1),F(x_2),\cdots,F(x_n)$。

注意到，$F(c)=F(x) \bmod (x\oplus c)$，于是可以考虑分治。

预处理出每个分治区间的 $x\oplus x_i$ 的乘积，每次将 F 关于两个子区间的乘积取模后分别递归处理即可。

时间复杂度为 $O\left(n\log^2 n\log\log n\right)$，使用转置原理可以得到一个常数更小的实现。

2.10 Nimber 多项式线性组合

给定 $2m$ 个多项式 $a_1(x),a_2(x),\cdots,a_m(x),b_1(x),b_2(x),\cdots,b_m(x)$，满足 $\deg b_i\le\deg a_i$。记 $n=\sum_{i=1}^{m}\deg b_i$，$A=\bigotimes_{i=1}^{m}a_i$

计算 $\bigoplus_{i=1}^{m}\frac{A}{a_i}\otimes b_i$。

实际上就是计算 $a_i(x)\oplus b_i(x)\otimes y$ 乘积中，y 一次项的系数。

在分治时保留 y^0,y^1 项系数即可。时间复杂度为 $O(n\log n\log m\log\log n)$。

2.11 Nimber 多项式插值

给定 $x_1,x_2,\cdots,x_n$ 和 $F(x_1),F(x_2),\cdots,F(x_n)$，求 $F(x)$。

考虑使用 Lagrange 插值公式：

[13]即利用给定的两个多项式线性组合出最大公因式

[14]即每次的商式或者说结果的 Bézout 系数

$$F(x)=\bigoplus_{i=1}^{n}\frac{\bigotimes_{j\neq i}(x\oplus x_j)}{\bigotimes_{j\neq i}(x_i\oplus x_j)}F(x_i)$$

注意到如果计算出了所有的分母 Q_i，就是一个线性组合问题了。

设 $M(x)=\bigotimes_{i=1}^{n}(x\oplus x_i)$，则第 i 个分子为 $P_i(x)=\frac{M(x)}{x\oplus x_i}$。

注意到 $M'(x)=\bigoplus_{i=1}^{n}P_i(x)$，于是 $Q_i=M'(x_i)$，做多点求值即可。

时间复杂度为 $O\left(n\log^2 n\log\log n\right)$。

2.12 Nimber 多项式中国剩余算法

给定 m 个互素的多项式 $a_1(x),a_2(x),\cdots,a_m(x)$，和 m 的多项式 $b_1(x),b_2(x),\cdots,b_m(x)$，求多项式 $F(x)$，使得 $F\equiv b_i\pmod{a_i}$。

记 $A=\bigotimes_{i=1}^{m}a_i$，$n=\deg M$。

由中国剩余定理，$F=\bigoplus_{i=1}^{m}b_i\left[\left(\frac{A}{a_i}\right)^{-1}\right]_{a_i}\frac{A(x)}{a_i}$

于是只需要算出所有的 $\left[\left(\frac{A}{a_i}\right)^{-1}\right]_{a_i}$，问题就变为了线性组合。

首先需要计算 $\frac{A}{a_i}\bmod a_i=\frac{A\bmod a_i^2}{a_i}$，类似多项式求值那样分治多项式取模即可。

然后计算 $\left[\left(\frac{A}{a_i}\right)^{-1}\right]_{a_i}$，即模多项式的求逆，使用欧几里得算法即可。

于是总的时间复杂度为 $O\left(n\log^2 n\log\log n\right)$。

2.13 **例2.** 小 Q 和小 Y 做游戏（二）

小 Q 和小 Y 是几何带师，这一次祂们找到了 n 个互不相同的非负整数，用来进行接下来 n 天的研究。

在第 i 天祂们准备研究 i 维的几何问题，于是祂们决定先来做一个游戏。

选出所有这 n 个数的所有 $\binom{n}{i}$ 个无序 i 元组，按照从小到大的顺序排列。则一个 i 元组可以看成 i 维空间中的一个点。

首先将这些 i 元组的点都染上黑色，其他点染上白色。小 Q 先手，轮流进行如下操作：

选择一个顶点都是非负整点、棱平行于坐标轴且坐标最大的点是黑点的 i 维长方体[15]，将它的所有顶点翻转颜色。如果选不出这样的长方体则当前操作者输。

由于小 Y 非常希望胜利，所以祂准备作弊。祂可以在小 Q 开始操作前，选择一个前 $i-1$ 维坐标都是 1 的非负整点并翻转这个点的颜色。

[15]这里要求长方体的 i 维体积为正

于是小 Y 准备寻求你的帮助，希望你求出，对于 $i = 1, 2, \cdots, n$，分别应该染哪个点。你只需要求最后一维的坐标。

$1 \le n \le 10^5, 0 \le a_i < 2^{32}$。

2.13.1 解法

不难发现，对于一个黑点，它的 SG 值就是每一维坐标的 Nim 积。

于是问题相当于计算 $\bigotimes_{i=1}^{n} (a_i \otimes x \oplus 1)$。

使用分治计算这些多项式的乘法即可。

时间复杂度 $O\left(n \log^2 n \log\log n\right)$。

3 Nimber 指数生成函数

3.1 Nimber 二项卷积

对 Nimber 多项式 a, b，其二项卷积定义为：

$$a \circledast b = \bigoplus_{i=0}^{n} \bigoplus_{j=0}^{m} \binom{i+j}{i} \cdot \left(a_i \otimes b_j \otimes x^{i+j}\right)$$

首先考虑 $\binom{i+j}{i}$ 模 2 的值，不难证明当且仅当 i, j 进行按位与运算为 0 时有 $\binom{i+j}{i}$ 模 2 为 1。

于是问题相当于计算 $\bigoplus_{i+j=k} [i \text{ and } j = 0] \left(a_i \otimes b_j\right)$。

注意到，在这个条件相当于 i, j 是 k 不相交的两个子集，所以也称为子集卷积。

下面我们定义集合幂级数，并给出子集卷积的计算方法。

3.1.1 Nimber 集合幂级数

对 $[0, 2^m)$ 的整数，我们可以将它理解为一个 $\{0, 1, \cdots, m-1\}$ 的子集，其 2^i 项系数决定了 i 是否在集合中。

于是一个多项式可以理解为一个集合幂级数。

首先定义集合幂级数的莫比乌斯变换：

$$\hat{f}_S = \bigoplus_{T \subseteq S} f_T$$

事实上，如果将集合看做一个 m 维向量，莫比乌斯变换就是在做 m 维的前缀和。

于是容易发现其逆变换就是 m 维的差分。由于在 Nimber 域中加减法相同，并且每维的长度就是 2，所以莫比乌斯变换的逆变换就是其本身。

对于高维前缀和，可以依次对每一维做前缀和。具体来说，设当前考虑第 i 维，则从小到大枚举每个位置，如果它在第 i 维有前驱就加上前驱的值。时间复杂度为 $O(m2^m) = O(n\log n)$。

莫比乌斯变换能解决集合并卷积问题：

$$f_S = \bigoplus_{L\cup R=S} g_L \otimes h_R$$

注意到，$\hat{f_S} = \bigoplus_{(L\cup R)\subseteq S} g_L \otimes h_R = \bigoplus_{L\subseteq S\wedge R\subseteq S} g_L \otimes h_R = \left(\bigoplus_{L\subseteq S} g_L\right) \otimes \left(\bigoplus_{R\subseteq S} h_R\right) = \hat{g}_S \otimes \hat{h}_S$，所以先对 g,h 做莫比乌斯变换，对应项相乘之后再做莫比乌斯变换即可。

从另一个角度考虑，集合并卷积是高维的 max 卷积，所以用高维前缀和处理是很合理的。

但我们现在需要做的并不是并卷积，而是子集卷积：

$$f_S = \bigoplus_{L\cup R=S\wedge L\cap R=\varnothing} g_L \otimes h_R$$

一个朴素的做法是，每次枚举所有的 $L \subseteq S$，则 $R = S \setminus L$。由于每个元素只可能是不在 S 中，在 L 中或在 R 中这三种情况，所以时间复杂度为 $O(3^m) = O\left(n^{\log_2 3}\right)$。

注意到 $[L \cup R = S \wedge L \cap R = \varnothing] = [L \cup R = S \wedge |L| + |R| = |S|]$。

于是给每个位置添加上一个多项式 $p(z)$，保证 $z^{|S|}$ 项系数为 f_S 且更低项的值为 0。这个多项式称为占位多项式。

则子集卷积就可以先求莫比乌斯变换，然后对应项的占位多项式做乘法，然后再做莫比乌斯变换。时间复杂度为 $O\left(m^2 2^m\right) = O\left(n\log^2 n\right)$

3.2 Nimber 半半在线二项卷积

定义 Nimber 半半在线二项卷积为：

$f = g \circledast h$，其中 g 已经给定且常数项为 0，而 h_i 需要等 f_i 计算完成后给出。

与下一个问题不同，这个问题是一个比较纯粹的集合幂级数问题。

定义 $\hat{f_{i,S}} = \bigoplus_{T\subseteq S\wedge|T|=i} f_T$。首先计算出所有的 $\hat{g_i}$。

考虑按照 $|S|$ 从 $0 \sim m$ 的顺序计算 f_S。

由于 $\hat{f_{i,S}} = \bigoplus_{j=1}^{i} \hat{g_{j,S}} \otimes \hat{h_{i-j,S}}$，计算的时间复杂度为 $O(m2^m) = O(n\log n)$。

然后进行一次莫比乌斯变换就能得到这一轮的 f_S，并得到相应的 h_S。再做莫比乌斯变换就能得到 $\hat{h_{i,S}}$。

总的时间复杂度为 $O\left(m^2 2^m\right) = O\left(n\log^2 n\right)$。

3.3 Nimber 全半在线二项卷积

定义 Nimber 全半在线二项卷积为：

$f = g \circledast h$，其中 g 已经给定，而 h_{i+1} 需要等 $f_0, f_1, \cdots, f_i$ 都计算完成后给出。

对于单纯的集合幂级数问题，不会出现这样的情况。但由于我们做的实际上是二项卷积，如果要解微分方程就可能会出现这样的情况。

首先考虑使用分治法，设当前处理的是 $[0, 2^n)$ 这段区间。

首先递归处理 $[0, 2^{n-1})$，然后计算 $[0, 2^{n-1})$ 的 h_i 对 $[2^{n-1}, 2^n)$ 的 f_i 进行的转移。

考虑递归右半部分，注意到此时 f, h 的最高位都是 1，所以可以直接转换成都是 $[0, 2^{n-1})$ 的情况。

时间复杂度为 $O\left(n \log^3 n\right)$。

事实上对于这个问题，直接使用 $O\left(n^{\log_2 3}\right)$ 的暴力法和分治效率差不多，甚至可能更快。

3.4 Nimber 指数生成函数复合的定义

在将指数生成函数的乘法解决后，下一个问题就是如何定义复合，首先我们要考虑的就是，如何不用除法定义两个指数生成函数的复合。

考虑计算 $\frac{f^k}{k!}$。

如果 f 不是单项式，就能拆成 $g + h$，然后做二项式展开就变成了 $\sum_{i=0}^{k} \frac{g^i \cdot h^{k-i}}{i! \cdot (k-i)!}$。

对于单项式，设 $f = \frac{ax^n}{n!}$，则结果就是 $\frac{x^{kn}}{(kn)!} \cdot \frac{(kn)!}{(n!)^k \cdot k!} \cdot a^k$。

不难证明 $\frac{(kn)!}{(n!)^k \cdot k!}$ 是一个整数，所以我们可以不进行除法定义两个指数生成函数的复合了。

仿照上面可以定义 Nimber 指数生成函数的复合，并给出了一个复杂度比较高的算法，实际上重要的是如何快速计算。

事实上，由于定义基本相同，所以照搬普通的指数生成函数的运算方法即可，即在积分的时候右移一位，求导的时候左移一位，多项式乘法用二项卷积。

需要注意的是，子集卷积只是求二项卷积的一种方法，这种方法省略了二项卷积中的一些项，而这些项会在复合中重新出现。所以直接对莫比乌斯变换后的占位多项式做复合的方法是错误的。

3.5 Nimber 二项卷积逆

不妨设常数项为 1，则二项卷积逆是多项式本身。

简单计算可以发现，交叉项都会出现两遍，而平方项只有常数项，所以平方后的结果就是 1。

而唯一性十分明显。

3.6 Nimber ln 与 Nimber exp

利用 ln 和 exp 的基本公式 $g' = g \circledast f'$，由于二项卷积逆是本身所以有 $g' \circledast g = f'$。

于是给定 g 求 f 只需要一次二项卷积。

考虑给定 f 求 g。由于这是一个微分方程，所以无法使用半半在线法解决，而全半在线的效率比较低。

事实上，这里可以使用牛顿迭代法。注意到 $\ln(1 \oplus x)$ 才是形式幂级数，所以方程变为了 $\ln(1 \oplus f) = g$。

带入牛顿迭代公式，得 $f_{t+1} = 1 \oplus (1 \oplus f_t) \circledast (1 \oplus \ln(1 \oplus f_t) \oplus g) \bmod x^{2^{t+1}}$。

时间复杂度同样和二项卷积相同。

3.7 Nimber 三角函数

通常三角函数计算方法需要对 2 做除法，无法在这里使用。

设 $g = \cos f$，则有 $g' = (e^f \oplus g) \circledast f'$。

两边乘上 e^f，得 $e^f \circledast g' = f' \oplus f' \circledast g \circledast e^f$，即 $\left(e^f \circledast g\right)' = f'$。

所以 $e^f \circledast g = f \oplus c$，令 $f = 0$ 得 $c = 1$。

于是 $\cos f = f \circledast e^f \oplus e^f$，$\sin f = f \circledast e^f$。

于是 $\tan f = \frac{\sin f}{\cos f} = \frac{f}{1+f} = f \circledast (1 + f) = f \oplus f^2 = f$。

3.8 **例3.** 小 Q 和小 Y 做游戏（三）

小 Q 和小 Y 都是图论带师，他们**特别**喜欢**完全图**。

对于一张图，如果它的每个连通块都是**完全图**，那么这张图是小 Q 和小 Y 所喜欢的，否则不喜欢。

现在，小 Q 和小 Y 在一些喜欢的图上做游戏。

1. 最开始时，每张图上大小为 i 的连通块上写着一个整数 f_i。
2. 在一个人操作时，首先选择一张图，满足这张图的每个连通块上写的数都是正整数。如果没有这样的图则当前操作者失败。
3. 对于每个连通块，选择一个比原来的数更小的非负整数。
4. 对于每个连通块集合的非空子集，在图的集合中添加一张选择图的拷贝，并把这张图在子集中的连通块上数改为新选择的数。
5. 将选择的图从图的集合中删去，并由另一个人从步骤 2 开始操作。

不难证明游戏的过程是有限的。

现在，小 Q 和小 Y 在所有喜欢的顶点数为 m 的带标号图进行上面的游戏，小 Q 先手。

由于小 Y 非常希望胜利，所以祂准备作弊。由于一些原因，祂只能修改 f_m 的值。

于是小 Y 准备寻求你的帮助，希望你求出，对于 $m = 1, 2, \cdots, n$，分别应该把 f_m 改成多少才能使得祂必胜。

$1 \le n \le 10^5, 0 \le f_i < 2^{32}$。

3.8.1 解法

对于一个图 G，容易发现它的 SG 值就是每个连通块上的数的 Nim 积。

注意到问题相当于在带标号的情况下把连通图变成多个连通块的组合，于是所有图的 SG 值就是 $\exp f$。

现在问要改成什么使得后手必胜，于是相当于要 Nim 和为 0，于是就是求 $f \oplus \exp f$。

使用上面介绍的 Nimber exp 算法即可。

时间复杂度为 $O\left(n \log^2 n\right)$。

总结

本文总结了 Nimber 上的多项式问题，给出了一些问题的解决方案。但对于全半在线二项卷积和指数生成函数复合没有得到好的做法。

同时，本文的一些算法[16]的实际应用，还有待开发。因此希望本文能够起到一个抛砖引玉的作用，希望感兴趣的读者，能够进行进一步研究，扩展笔者做法，从而得到更多更有趣的做法与应用。

感谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队高闻远教练的指导。

感谢父母对我的培养和教育。

感谢学校的栽培，符水波老师和应平安老师的教导和同学们的帮助。

感谢虞皓翔同学，胡家齐同学，钱易同学与我交流讨论、给我启发。

感谢翁伟捷同学，潘佳奇同学，宣毅鸣同学为本文审稿。

[16]如三角函数

参考文献

[1] John H. Conway. ”On numbers and games.” (2001).

[2] H.W. Lenstra. ”Nim multiplication.” Séminaire De Théorie Des Nombres 11(1978).

[3] Wikipedia, the free encyclopedia, “Sprague–Grundy theorem”,https://en.wikipedia.org/wiki/Sprague–Grundy_theorem.

[4] Wikipedia, the free encyclopedia, “Nimber”,https://en.wikipedia.org/wiki/Nimber.

[5] 彭雨翔，《多项式导论》，2016 年冬令营。

[6] 吕凯风，《集合幂级数的性质及其快速算法》，IOI2015中国国家队候选队员论文集。

[7] 毛啸, 《再探快速傅里叶变换》, IOI2016中国国家队候选队员论文集。