集合幂级数的性质与应用及其快速算法

武汉市第二中学 吕凯风

摘 要

为了研究以集合为状态的递推，本文提出了集合幂级数，并分析了集合幂级数的性质，总结归纳了三种常见的乘法及相关快速算法。最后本文给出了几个利用集合幂级数加速递推的实际例子。

1 引言

在信息学竞赛中，经常会遇到一些有关集合的动态规划问题。比如是某种关于集合的计数问题，通常你会设一个 f_S 表示对于集合 S 的某某方案数，然后写一个递推式。等你写完了递推式，看了看题目规定的时间限制——这个算法太慢了。这个时候，怎么办？你想要优化递推式，但是手上却没有称手的工具，那你或许就只能选择放弃了。

回忆我们较为成熟的数列理论，我们可以把一个数列 $f_0, f_1, \ldots$ 和一个形式幂级数 $f(x) = \sum_{k\geq 0} f_k x^k$ 相对应，并称之为该数列的生成函数。由于左有数学上对形式幂级数的缜密分析，右有计算上对形式幂级数的快速算法，所以生成函数一马平川，解决了一个又一个数列的计算问题。

不禁想问，我们是否能把 f_S 这样的下标为集合的数列也跟一个幂级数挂钩并作为其生成函数，从而产生优美而高效的算法呢？本文作者做了一番研究，得到了这篇论文要展示的内容。

在本文第 2 节，本文作者给出了集合幂级数的定义以及加法运算，讨论了应该如何定义乘法。

接下来第 3, 4, 5 节兵分三路，从三种途径定义乘法并介绍相关快速算法，进一步分析性质。

在第 6 节，本文作者给出了几个集合幂级数的实际应用。

为了方便，我们规定如下几个本文中要用到的记号。

记号 1.1. 我们设全集为有限集 $U = \{1, 2, \ldots, n\}$，其中 $n = |U|$。本文中我们假设所有我们关心的 f_S 中的集合 S 都是 U 的子集。

记号 1.2. 设 X 是一个集合，用 2^X 表示 X 的幂集，即 X 的所有子集组成的集合。

记号 1.3. 用 $[P]$ 表示表达式 P 成立时为 1 否则为 0。例如 $[x = 3]$ 表示 $x = 3$ 时值为 1 否则为 0。

2 定义

现在我们引入集合幂级数。

定义 2.1. 设 F 是一个域，则称函数 $f : 2^U \to F$ 为 F 上的一个**集合幂级数**。对每个 $S \in 2^U$，记 f_S 表示把 S 代入函数 f 后的函数值，并称 f_S 为该集合幂级数的第 S 项的**系数**。显然如果确定了每一个系数的值，那么 f 也就确定了。

记号 2.1. 常见的域有 $\mathbb{C}, \mathbb{R}, \mathbb{Q}, \mathbb{F}_p$ 等等，域具有基本相同的性质。本文中设我们要研究的是 F 上的集合幂级数，而不是一个个具体分析每个域上的集合幂级数的情况。所以本文中不明确指出时，我们都用 F 表示要研究的集合幂级数的系数所属的域。

集合幂级数之间的加法运算是容易定义的。

定义 2.2. 设 f, g 为集合幂级数，定义 $f + g = h$，其中 h 是一个集合幂级数，且 h_S 满足：

$$h_S = f_S + g_S \tag{1}$$

即对应系数相加。

减法只要把 $f_S + g_S$ 改为 $f_S - g_S$ 就行了。显见，我们可以在 $O(2^n)$ 时间内求两个集合幂级数的加减法。

容易看出集合幂级数形成了一个加法阿贝尔群，其中零元为系数均为 0 的集合幂级数。

记号 2.2. 对于任意 $c \in F$，$S \in 2^U$，我们用符号 $f = cx^S$ 来表示一个第 S 项系数为 c，其余项均为 0 的集合幂级数 f。这里的 cx^S 是一个整体的符号，并不是某种乘法。按照习惯，对于 $1x^S$ 我们可以省略为 x^S。

容易看出一个集合幂级数一定可以写成若干个 cx^S 相加的形式。于是我们可以用符号：

$$f = \sum_{S \subseteq 2^U} f_S x^S \tag{2}$$

来表示一个集合幂级数。

例如取 $U = \{1, 2\}$ 以及 $F = \mathbb{R}$，我们用 $f(x) = 5x^{\varnothing} + 9x^{\{1\}} + 3x^{\{1,2\}}$ 可以表示一个 $f_{\varnothing} = 5, f_{\{1\}} = 9, f_{\{2\}} = 0, f_{\{1,2\}} = 3$ 的集合幂级数。

那么现在问题来了，乘法应该如何定义？

记乘法为 $h = f \cdot g$，显然我们希望乘法对加法有分配律，那么：

$$\sum_{S \in 2^U} h_S x^S = \left(\sum_{L \in 2^U} f_L x^L\right) \cdot \left(\sum_{R \in 2^U} g_R x^R\right) = \sum_{L \in 2^U} \sum_{R \in 2^U} \left(f_L x^L\right) \cdot \left(g_R x^R\right) \tag{3}$$

自然我们希望 $\left(f_L x^L\right) \cdot \left(g_R x^R\right)$ 是以某种集合运算乘起来的，我们设一个 2^U 里的二元运算 $*$，满足：

- $\forall L, R \in 2^U$：$L * R = R * L$（交换律）
- $\forall L, M, R \in 2^U$：$(L * M) * R = L * (M * R)$（结合律）
- $\forall S \in 2^U$：$S * \varnothing = S$（单位元为 $\varnothing$）

那么我们就可以定义 $\left(f_L x^L\right) \cdot \left(g_R x^R\right) = (f_L g_R) x^{L*R}$，易证这样定义集合幂级数的乘法满足交换律、结合律、对加法的分配律。我们可以把所有 $c \in F$ 的元素看作 $cx^{\varnothing}$，这样能使得对于任意集合 $S \in 2^U$ 有 $cx^S = c \cdot x^S$。于是集合幂级数形成了一个交换环，并且包含了整个 F 作为子环。

把 $*$ 取成不同的运算可以得到不同的效果。下面三节我们将分别看到由不同的 $*$ 定义产生的集合并卷积、集合对称差卷积、子集卷积三种乘法。

3 集合并卷积

我们取 $L * R = L \cup R$ 就可以得到集合并卷积。

定义 3.1. 我们定义两个集合幂级数的乘积为**集合并卷积**。设 $f = \sum_{S \subseteq 2^U} f_S x^S$，$g = \sum_{S \subseteq 2^U} g_S x^S$，定义 $f \cdot g = h$，其中 h 是一个集合幂级数，且 h_S 满足：

$$h_S = \sum_{L \subseteq 2^U} \sum_{R \subseteq 2^U} [L \cup R = S] f_L g_R \tag{4}$$

为了方便，$f \cdot g$ 可以简记为 fg。

3.1 快速算法

给出两个集合幂级数，暴力枚举 L 和 R 把 $f_L g_R$ 加到 $h_{L\cup R}$ 上去，这样进行集合并卷积的计算的时间复杂度是 $O(4^n)$，通常是无法接受的。

我们需要更快的算法。

3.1.1 分治乘法

我们考虑 U 中的元素 n。我们可以把含 n 的集合单独提一个 $\{n\}$ 出来，那么就可以写成 $f = f^- + x^{\{n\}} f^+, g(x) = g^- + x^{\{n\}} g^+$ 的形式。于是：

$$\begin{aligned} fg &= \left(f^- + x^{\{n\}} f^+\right)\left(g^- + x^{\{n\}} g^+\right) && (5)\\ &= f^- g^- + x^{\{n\}}(f^- g^+ + f^+ g^- + f^+ g^+) && (6)\\ &= f^- g^- + x^{\{n\}}((f^- + f^+)(g^- + g^+) - f^- g^-) && (7) \end{aligned}$$

所以我们把问题转化成了求 $f^- g^-$ 和 $(f^- + f^+)(g^- + g^+)$，而这里的集合幂级数的集合中都不含 n 这个元素了，所以可以令 $U = \{1, 2, \ldots, n-1\}$ 然后递归进行乘法即可。

设时间复杂度为 $T(n)$ 那么 $T(n) = 2T(n-1) + O(2^n)$，解得 $T(n) = O(n2^n)$。

3.1.2 快速莫比乌斯变换

我们来看一个更加强大的计算乘法的算法。

定义 3.2. 对于一个集合幂级数 f 我们定义 f 的**莫比乌斯变换**[1]为集合幂级数 $\hat{f}$，其中：

$$\hat{f}_S = \sum_{T\subseteq S} f_T \tag{8}$$

反过来，我们定义 $\hat{f}$ 的**莫比乌斯反演**[2]为 f。

由容斥原理，我们可以得到：

$$f_S = \sum_{T\subseteq S} (-1)^{|S|-|T|} \hat{f}_T \tag{9}$$

[1]Möbius Transform
[2]Möbius Inversion

所以对于任意一个集合幂级数，莫比乌斯反演是存在且唯一的。

我们对 (4) 等式两端同时做莫比乌斯变换，那么就可以得到：

$$\hat{h}_S = \sum_{L\subseteq 2^U}\sum_{R\subseteq 2^U}[L\cup R\subseteq S]f_L g_R \tag{10}$$

我们知道，由于 $[L\cup R\subseteq S]=[L\subseteq S][R\subseteq S]$ 所以：

$$\begin{aligned}\hat{h}_S &= \sum_{L\subseteq S}\sum_{R\subseteq S} f_L g_R &(11)\\ &= \left(\sum_{L\subseteq S} f_L\right)\left(\sum_{R\subseteq S} g_R\right) &(12)\\ &= \hat{f}_S\hat{g}_S &(13)\end{aligned}$$

所以，如果我们想求 $h=fg$，我们可以先求出 $\hat{f}$ 和 $\hat{g}$，对应系数乘起来得到 $\hat{h}$，再通过 $\hat{h}$ 求 h。那么现在关键问题在于，怎样快速求出一个集合幂级数的莫比乌斯变换以及莫比乌斯反演。

我们可以使用递推。设 $\hat{f}_S^{(i)}$ 为 $S\setminus T\subseteq\{1,\dots,i\}$ 的 S 的子集 T 的系数之和。令 $\hat{f}_S^{(0)}=f_S$，容易得到对于每个不包含 i 的 S 有 $\hat{f}_S^{(i)}=\hat{f}_S^{(i-1)}$，$\hat{f}_{S\cup\{i\}}^{(i)}=\hat{f}_{S\cup\{i\}}^{(i-1)}+\hat{f}_S^{(i-1)}$。

我们可以用同样的方法计算莫比乌斯反演。两个算法的时间复杂度都是 $O(n2^n)$。下面给出了代码示例：

Algorithm 1 快速莫比乌斯变换

输入： 集合幂级数 f

输出： f 的莫比乌斯变换

```
for i ← 1 to n do
    for all S ∈ 2^{U\{i}} do
        f_{S∪{i}} ← f_{S∪{i}} + f_S
    end for
end for
return f
```

Algorithm 2 快速莫比乌斯反演

输入： 集合幂级数 f

输出： f 的莫比乌斯反演

```
for i ← 1 to n do
    for all S ∈ 2^{U\{i}} do
        f_{S∪{i}} ← f_{S∪{i}} − f_S
    end for
end for
return f
```

这样，我们就能在 $O(n2^n)$ 时间内计算两个集合幂级数的乘法。

3.1.3 比较

我们比较一下这两种算乘法的算法。两个算法时间复杂度相同。仔细观察发现分治乘法其实是把莫比乌斯变换与莫比乌斯反演一起进行了——递归下去的时候是在做莫比乌斯变换，回溯上来的时候是在做莫比乌斯反演。这两个算法的本质是一样的。

值得一提的是，这两种算法都能拓展到任意交换环 R 上计算 R 上的集合幂级数的乘法。

3.2 由变换引出的性质

当我们把乘法拆成做莫比乌斯变换，对应系数相乘，做莫比乌斯反演这三个过程的时候，如果我们要把三个集合幂级数乘起来，只要分别求莫比乌斯变换，把对应系数乘起来之后再做莫比乌斯反演就行了，而不用分别做两次乘法。

这个结论启发我们把某些对集合幂级数的操作转化为对莫比乌斯变换的操作。由于 $f+g$ 的莫比乌斯变换的系数等于 $\hat{f}$ 和 $\hat{g}$ 的对应系数相加，fg 的莫比乌斯变换的系数等于 $\hat{f}$ 和 $\hat{g}$ 的对应系数相乘，所以对于一个集合幂级数 f，如果有一个多项式 $T(x)$，我们想求 $T(f)$，那么我们只用算出 $\hat{f}$，然后对每个系数 $\hat{f}_S$ 求出 $T(\hat{f}_S)$，这样就得到了 $T(f)$ 的莫比乌斯变换，对其做莫比乌斯反演就能得到 $T(f)$。

很自然地，我们会想知道能否做除法。作为乘法的逆运算，容易看出做除法就是对莫比乌斯变换的系数做除法。所以，一个集合幂级数有唯一的乘法逆元当且仅当莫比乌斯变换的所有系数都不为 0。

上述内容看起来似曾相识。我们常用快速离散傅里叶变换在 $O(n\log n)$ 的时间内计算两个多项式的乘法，而这个算法既可以像上文那样抽象地推导出来，也可以形象地理解成把所有 n 次单位根代入多项式求值，把函数值乘起来，再用插值求出多项式的系数。

我们可以尝试得出一个快速莫比乌斯变换的另一个版本的理解。

定义 3.3. 对于一个 n 维向量 $\mathbf{x}\in F^n$ 和一个集合 $S\in 2^U$，我们定义：

$$\mathbf{x}^S=\prod_{i\in S}\mathbf{x}_i \tag{14}$$

定义 3.4. 对于一个 n 维向量 $\mathbf{x}\in F^n$ 和一个集合幂级数 f，我们定义：

$$f(\mathbf{x})=\sum_{S\subseteq 2^U} f_S\cdot\mathbf{x}^S \tag{15}$$

可以看出 $f(\mathbf{x})$ 是一个关于 $\mathbf{x}_1,\ldots,\mathbf{x}_n$ 的 n 元多项式函数。

定义 3.5. 对于一个集合 $S\in 2^U$，定义 S 的**特征向量**为 $\mathbf{S}\in F^n$，其中：

$$\mathbf{S}_i=[i\in S] \tag{16}$$

那么我们可以发现，$\hat{f_S}=f(\mathbf{S})$，所以莫比乌斯变换就是把所有集合的特征向量代入集合幂级数求值，莫比乌斯反演就是插值。注意到 0 和 1 这两个数的任意正整数次幂都等于它本身，所以对于一个 01 向量 $\mathbf{x}$ 必有 $\mathbf{x}^L\cdot\mathbf{x}^R=\mathbf{x}^{L\cup R}$。所以对于 $h=fg$ 我们就自然而然地有 $h(\mathbf{S})=f(\mathbf{S})g(\mathbf{S})$。这样我们就从另一个视角解释了这个计算乘法的算法。

4 集合对称差卷积

接下来我们换一种方式定义乘法。

定义 4.1. 定义两个集合 A,B 的**对称差**为：

$$A\oplus B=\{x\mid (x\in A)\oplus(x\in B)\} \tag{17}$$

其中右侧的 $\oplus$ 为逻辑异或，即两个表达式真假性不同时为真，否则为假。

我们取 $L * R = L \oplus R$ 就可以得到集合对称差卷积。

定义 4.2. 我们定义两个集合幂级数的乘积为**集合对称差卷积**。设 $f = \sum_{S \subseteq 2^U} f_S x^S$，$g = \sum_{S \subseteq 2^U} g_S x^S$，定义 $f \cdot g = h$，其中 h 是一个集合幂级数，且 h_S 满足：

$$h_S = \sum_{L \subseteq 2^U} \sum_{R \subseteq 2^U} [L \oplus R = S] f_L g_R \tag{18}$$

为了方便，$f \cdot g$ 可以简记为 fg。

4.1 快速算法

给出两个集合幂级数，暴力枚举 L 和 R 把 $f_L g_R$ 加到 $h_{L \oplus R}$ 上去，这样进行集合对称差卷积的计算的时间复杂度是 $O(4^n)$，通常是无法接受的。

我们需要更快的算法。

4.1.1 分治乘法

跟集合并卷积的时候一样，我们还是可以使用分治乘法。我们考虑 U 中的元素 n。我们可以把含 n 的集合单独提一个 $\{n\}$ 出来，那么就可以写成 $f = f^- + x^{\{n\}} f^+, g(x) = g^- + x^{\{n\}} g^+$ 的形式。于是：

$$\begin{aligned} fg &= \left(f^- + x^{\{n\}} f^+\right)\left(g^- + x^{\{n\}} g^+\right) && (19) \\ &= (f^- g^- + f^+ g^+) + x^{\{n\}}(f^- g^+ + f^+ g^-) && (20) \end{aligned}$$

我们可以转而求 $(f^- + f^+)(g^- + g^+)$ 和 $(f^- - f^+)(g^- - g^+)$，然后两式相加除以 2 得到 $f^- g^- + f^+ g^+$，两式相减除以 2 得到 $f^- g^+ + f^+ g^-$。我们可以令 $U = \{1, 2, \ldots, n-1\}$ 然后递归进行乘法即可。

由于涉及除以 2，该算法不能对特征为 2 的 F 使用。

设时间复杂度为 $T(n)$ 那么 $T(n) = 2T(n-1) + O(2^n)$，解得 $T(n) = O(n2^n)$。

4.1.2 快速沃尔什变换

我们来看一个更加强大的计算乘法的算法——快速沃尔什变换。快速沃尔什变换其实也就是每一维大小为 2 的高维快速傅里叶变换。

原算法是用矩阵的语言描述的，而下面我们用集合的语言来描述。

首先注意到对于一个集合 S 有：

$$\frac{1}{2^n}\sum_{T\subseteq 2^U}(-1)^{|S\cap T|}=[S=\varnothing] \tag{21}$$

这只用注意到，当 S 为空集时，$(-1)^{|S\cap T|}$ 恒为 1。当 S 不为空时，设 $v\in S$，那么考虑一个 $T\in 2^U$，由于 $(-1)^{|S\cap(T\oplus\{v\})|}=(-1)^{|(S\cap T)\oplus\{v\}|}=-(-1)^{|S\cap T|}$，所以 $(-1)^{|S\cap T|}+(-1)^{|S\cap(T\oplus\{v\})|}=0$。由于 $T\oplus\{v\}\oplus\{v\}=T$，所以我们可以把每个 T 跟 $T\oplus\{v\}$ 配成一对，所以当 S 不为空时，原式左边为 0。

利用这个性质，我们可以化简 (18)：

$$\begin{aligned}
h_S &= \sum_{L\subseteq 2^U}\sum_{R\subseteq 2^U}[L\oplus R\oplus S=\varnothing]f_Lg_R &(22)\\
&= \sum_{L\subseteq 2^U}\sum_{R\subseteq 2^U}\frac{1}{2^n}\sum_{T\subseteq 2^U}(-1)^{|T\cap(L\oplus R\oplus S)|}f_Lg_R &(23)\\
&= \sum_{L\subseteq 2^U}\sum_{R\subseteq 2^U}\frac{1}{2^n}\sum_{T\subseteq 2^U}(-1)^{|T\cap L|}(-1)^{|T\cap R|}(-1)^{|T\cap S|}f_Lg_R &(24)\\
&= \frac{1}{2^n}\sum_{T\subseteq 2^U}(-1)^{|T\cap S|}\left(\sum_{L\subseteq 2^U}(-1)^{|T\cap L|}f_L\right)\left(\sum_{R\subseteq 2^U}(-1)^{|T\cap R|}g_R\right) &(25)
\end{aligned}$$

这就启发我们定义沃尔什变换：

定义 4.3. 对于一个集合幂级数 f 我们定义 f 的**沃尔什变换**[1]为集合幂级数 $\hat{f}$，其中：

$$\hat{f}_S=\sum_{T\subseteq 2^U}f_T(-1)^{|S\cap T|} \tag{26}$$

反过来，我们定义 $\hat{f}$ 的**沃尔什逆变换**[2]为 f。

容易用和上面类似的推理发现沃尔什逆变换是唯一的（事实上只用取 $g=x^{\varnothing}$ 即可），且为：

$$f_S=\frac{1}{2^n}\sum_{T\subseteq 2^U}\hat{f}_T(-1)^{|S\cap T|} \tag{27}$$

我们再看集合对称差卷积，就有：

$$\hat{h}_S=\hat{f}_S\hat{g}_S \tag{28}$$

[1]Walsh Transform

[2]Inverse Walsh Transform

于是如果想求 $h = fg$，我们可以先对 f, g 分别做沃尔什变换，对应系数乘起来得到 $\hat{h}$，再做沃尔什逆变换。所以现在问题变成了，怎样快速求一个集合幂级数的沃尔什变换以及沃尔什逆变换。

我们可以使用递推。设 $\hat{f}_S^{(i)}$ 为只考虑那些与 S 的对称差是 $\{1, \ldots, i\}$ 的子集的集合时的沃尔什变换第 S 项，即那些 $S \oplus T \subseteq \{1, \ldots, i\}$ 的集合 T 的 $f_T(-1)^{|S \cap T|}$ 之和。令 $\hat{f}_S^{(0)} = f_S$，容易得到对于每个不包含 i 的 S 有 $\hat{f}_S^{(i)} = \hat{f}_S^{(i-1)} + \hat{f}_{S \cup \{i\}}^{(i-1)}$ 且 $\hat{f}_{S \cup \{i\}}^{(i)} = \hat{f}_S^{(i-1)} - \hat{f}_{S \cup \{i\}}^{(i-1)}$。

我们可以做沃尔什变换后乘以 $\frac{1}{2^n}$ 得到沃尔什逆变换。于是做沃尔什变换和沃尔什逆变换的时间复杂度都是 $O(n2^n)$。下面给出了代码示例：

Algorithm 3 快速沃尔什变换

输入： 集合幂级数 f

输出： f 的沃尔什变换

```
for i ← 1 to n do
    for all S ∈ 2^{U\{i}} do
        l ← f_S, r ← f_{S∪{i}}
        f_S = l + r, f_{S∪{i}} = l − r
    end for
end for
return f
```

这样，我们就能在 $O(n2^n)$ 时间内计算两个集合幂级数的乘法。

不过遗憾的是，由于涉及乘以 $\frac{1}{2^n}$，该算法不能对特征为 2 的 F 使用。

4.1.3 比较

我们比较一下这两种算乘法的算法。仔细观察发现分治乘法其实是把沃尔什变换与沃尔什逆变换一起进行了——递归下去的时候是在做沃尔什变换，回溯上来的时候是在做沃尔什逆变换。这两个算法的本质是一样的。

然而两个算法都无法对特征为 2 的 F 使用，这是一个小小的遗憾。实际上两个算法都能对任意 2 有乘法逆元的交换环上的集合幂级数使用。

4.2 由变换引出的性质

跟集合并卷积一样，当我们把乘法拆成做沃尔什变换，对应系数相乘，做沃尔什逆变换这三个过程的时候，如果我们要把三个集合幂级数乘起来，只要分别求沃尔什变换，把对应系数乘起来之后再做沃尔什逆变换就行了，而不用分别做两次乘法。

这个结论又启发我们把某些对集合幂级数的操作转化为对沃尔什变换的操作。由于 $f+g$ 的沃尔什的系数等于 $\hat{f}$ 和 $\hat{g}$ 的系数相加，fg 的沃尔什变换的系数等于 $\hat{f}$ 和 $\hat{g}$ 的系数相乘，所以对于一个集合幂级数 f，如果有一个多项式 $T(x)$，我们想求 $T(f)$，那么我们只用算出 $\hat{f}$，然后对每个系数 $\hat{f}_S$ 求出 $T(\hat{f}_S)$ 得到 $T(f)$ 的沃尔什变换，对其做沃尔什逆变换就能得到 $T(f)$。

很自然地，我们又会想知道能否做除法。作为乘法的逆运算，容易看出做除法就是对沃尔什变换的系数做除法。所以，一个集合幂级数有唯一的乘法逆元当且仅当沃尔什变换的所有系数都不为 0。

我们可以尝试得出一个快速沃尔什变换的另一个版本的理解。

我们沿用集合并卷积一节中 $\mathbf{x}^S$ 和 $f(\mathbf{x})$ 的定义，重新定义集合的特征向量。

定义 4.4. 对于一个集合 $S \in 2^U$，定义 S 的**特征向量**为 $\mathbf{S} \in F^n$，其中：

$$\mathbf{S}_i = (-1)^{[i \in S]} \tag{29}$$

那么我们可以发现，$\hat{f}_S = f(\mathbf{S})$，所以沃尔什变换就是把所有集合的特征向量代入集合幂级数求值，沃尔什逆变换就是插值。注意到 ± 1 的平方都等于 1，所以对于一个每一维均为 ± 1 的向量 $\mathbf{x}$ 必有 $\mathbf{x}^L \cdot \mathbf{x}^R = \mathbf{x}^{L \oplus R}$。所以对于 $h = fg$ 我们就自然而然地有 $h(\mathbf{S}) = f(\mathbf{S})g(\mathbf{S})$。这样我们就从另一个视角解释了这个计算乘法的算法。

5 子集卷积

接下来我们再换一种方式定义乘法。这次我们定义 $*$ 为不相交集合并，即如果 $A \cap B \neq \varnothing$ 那么 $A * B$ 未定义（严谨地说，$x^A \cdot x^B = 0$），否则为 $A \cup B$。下面是正式的定义。

定义 5.1. 我们定义两个集合幂级数的乘积为**子集卷积**[1]。设 $f = \sum_{S \subseteq 2^U} f_S x^S$，

[1] subset convolution

$g = \sum_{S\subseteq 2^U} g_S x^S$，定义 $f \cdot g = h$，其中 h 是一个集合幂级数，且 h_S 满足：

$$h_S = \sum_{L\subseteq S}\sum_{R\subseteq S}[L \cup R = S][L \cap R = \varnothing] f_L g_R \tag{30}$$

为了方便，$f \cdot g$ 可以简记为 fg。

5.1 快速算法

给出两个集合幂级数，暴力枚举集合 S 和子集 L，可以得到 $R = S \setminus L$，把 $f_L g_R$ 加到 h_S 上去，这样进行子集卷积的计算的时间复杂度是 $O(3^n)$ 的。这是因为 $\sum_{i=0}^{n}\binom{n}{i}2^i = 3^n$。

下面我们考虑更快的算法。

5.1.1 转化为集合并卷积

注意到 $[L \cup R = S][L \cap R = \varnothing] = [L \cup R = S][|L| + |R| = |S|]$，这就启发我们对 F 上的形式幂级数环 $F[[z]]$ 定义集合幂级数。通过增加未知数 z，我们可以用 z 的次数表示集合大小来限制 $|L| + |R| = |S|$。

对于一个 $F[[z]]$ 上的集合幂级数 σ，我们仍旧用符号 σ_S 表示 σ 的第 S 项的系数，但此时这个系数是个 F 上的形式幂级数。我们用 $\sigma_{S,i}$ 表示 σ_S 的第 i 项的系数。

定义 5.2. 我们称一个 $F[[z]]$ 上的集合幂级数 σ 为集合幂级数 f 的**集合占位幂级数**当且仅当对于任意一个集合 $S \in 2^U$，对于 $0 \le i < |S|$ 满足 $\sigma_{S,i} = 0$，且 $\sigma_{S,|S|} = f_S$。即：

$$\sigma = \sum_{S\in 2^U} f_S z^{|S|} x^S + \sum_{S\in 2^U}\sum_{k=|S|+1}^{\infty} \sigma_{S,i} z^i x^S \tag{31}$$

现在我们回到我们要研究的子集卷积。现在事情就很简单了，要把 f, g 给乘起来，我们就分别取它们的集合占位幂级数 σ, τ，然后求 σ, τ 的集合并卷积得到 δ，那么必有 $h_S = \delta_{S,|S|}$。由于形式幂级数加减是 $O(n)$ 的，相乘是 $O(n^2)$ 的，所以算法的时间复杂度为 $O(n^2 2^n)$。

把中间的集合并卷积的快速算法展开来看，算法就变成了：对集合占位幂级数 σ, τ 做莫比乌斯变换，对应系数乘起来，此处是个形式幂级数的乘法，然后再做莫比乌斯反演得到 δ。

5.1.2 转化为集合对称差卷积

注意到 $[L \cup R = S][L \cap R = \varnothing] = [L \oplus R = S][|L| + |R| = |S|]$，所以我们也可以取 f, g 的集合占位幂级数 σ, τ 然后求 σ, τ 的集合对称差卷积得到 δ，那么必有 $h_S = \delta_{S,|S|}$。算法的时间复杂度也是 $O(n^2 2^n)$。

5.1.3 比较

比较上述两种算法，由于集合对称差卷积的快速算法需要 2 有逆元，略有局限性，而实际上任意交换环上的集合幂级数都可以快速求集合并卷积，所以本文作者更推荐使用集合并卷积计算子集卷积。

5.2 由变换引出的性质

无论是用集合并卷积还是用集合对称差卷积，我们都把乘法变成了变换、系数相乘、逆变换，我们就可以把对某些集合幂级数的操作转化为对集合占位幂级数的变换的系数的操作，再做逆变换。

我们再来考虑除法。显然一个形式幂级数有乘法逆元当且仅当常数项不为 0，所以一个集合幂级数有乘法逆元当且仅当第 $\varnothing$ 项系数不为 0。我们可以用递推在 $O(n^2)$ 时间内求出形式幂级数的逆元的 0 到 n 次项，所以也就能在 $O(n^2 2^n)$ 时间内求出集合幂级数的乘法逆元。

我们还可以定义集合幂级数的导数，$(cx^S)' = \sum_{v \in S} cx^{S \setminus \{v\}}$。可以证明这样定义满足导数的性质。但是另一个方面，定义积分是很困难的。

在后面的应用中我们可以看得出，子集卷积比集合并卷积、集合对称差卷积更有组合意义。

6 应用

接下来我们来看看集合幂级数的一些应用。

6.1 随机集合并为全集的期望集合数

6.2 描述

设 $U = \{1, \ldots, n\}$。有一个人在玩游戏机。每回合，游戏机会随机产生一个 U 的子集，其中产生子集 S 的概率为 p_S。当游戏机产生的集合的并集为 U 时游戏结束。

给你 n 和每个集合 S 产生的概率 p_S，问期望多少回合游戏结束。

数据范围：$n \leq 20$。

6.2.1 解法

我们把 p 看成集合幂级数，那么第 k 回合结束后并集为 S 的概率为集合幂级数 p^k 的第 S 项。我们用集合并卷积来定义乘法。

那么游戏在第 k 回合结束的概率为 $p^k - p^{k-1}$ 的第 U 项。所以期望回合数就是下面这个集合幂级数的第 U 项：

$$\sum_{k=1}^{\infty} k(p^k - p^{k-1})$$

设这个集合幂级数为 f。

做莫比乌斯变换，那么：

$$\hat{f}_S = \sum_{k=1}^{\infty} k(\hat{p}_S^k - \hat{p}_S^{k-1})$$

由于概率是 $[0, 1]$ 中的实数且和为 1，所以 $0 \leq \hat{p}_S \leq 1$。所以：

$$\hat{f}_S = \begin{cases} -\frac{1}{1-\hat{p}_S} & \hat{p}_S < 1 \\ 0 & \hat{p}_S = 1 \end{cases}$$

然后对 $\hat{f}$ 做莫比乌斯反演得到 f 即可。

如果并集永远不可能为 U 那么 p^k 的第 U 项必为 0，所以这样得到的 f_U 也是 0。

所以我们就得到了时间复杂度 $O(n2^n)$ 的优秀算法。

6.3 Topcoder SRM 518 Nim

6.4 描述

两个人在玩 Nim 游戏，有 m 堆石子，每次可以选一堆石子拿走若干个石子，不能不拿。不能拿的人输。

求每堆石子数为不超过 l 的素数且先手必败的方案数对 10^9+7 取模后的结果。

数据范围：$m \le 10^9, l \le 10^5$。

6.5 解法

由斯普莱格 - 格隆第定理[1]知，先手必败当且仅当石子数的异或和为 0。

集合可以跟二进制数相对应，对称差就是异或。用集合对称差卷积来定义乘法，用 set(k) 表示 k 这个二进制数对应的集合，那么这个问题就是求：

$$\left(\sum_{p=1}^{l}[p\text{是素数}]x^{\text{set}(p)}\right)^m \tag{32}$$

的第 $\varnothing$ 项系数。

这里需要对一个集合幂级数求 m 次方。我们可以先对这个集合幂级数做沃尔什变换，然后用快速幂求每一个系数的 m 次方，然后再做沃尔什逆变换即可。

所以我们就得到了时间复杂度 $O(l\log l + l\log m)$ 的优秀算法。

6.6 连通生成子图计数

6.7 描述

给你一个 n 个结点的无向图 G，求连通生成子图的个数对 10^9+7 取模后的结果。

G 的生成子图即包含 G 中所有结点的子图。

数据范围：$n \le 20$。

[1] Sprague - Grundy theorem

6.8 解法

我们设 f_S 为 G 由结点集 S 导出的子图的连通生成子图的个数，g_S 为由结点集 S 导出的子图的生成子图的个数。特别地，令 $f_\varnothing = g_\varnothing = 0$。

把 f, g 看成集合幂级数，用子集卷积来定义乘法，则：

$$1 + g = \sum_{k\geq 0} \frac{f^k}{k!}$$

右边就是 e^f 的幂级数形式，所以我们可以写成：

$$1 + g = e^f$$

于是可以解得：

$$f = \ln(1 + g) = \sum_{k\geq 1} \frac{(-1)^{k+1}}{k} g^k$$

g 是好求的，只要求出每个导出子图的边数 m_S 然后对于 $S \neq \varnothing$ 就有 $g_S = 2^{m_S}$。所以我们可以求出 g 的一个集合占位幂级数 σ 然后做莫比乌斯变换得到 $\hat{\sigma}$，然后对于每个 $\hat{\sigma}_S$ 求 $\ln(1 + \hat{\sigma}_S)$，这里是对形式幂级数取自然对数，最后再做莫比乌斯反演得到 f 的一个集合占位幂级数。

对于形式幂级数 $a = \sum_{k\geq 1} a_k z^k$ 求 $b = \ln(1 + a)$ 的方法：

$$\begin{aligned}
b' &= \frac{a'}{1 + a} \\
b'(1 + a) &= a' \\
b'_k + \sum_{i=0}^{k-1} b'_i a_{k-i} &= a'_k \\
(k + 1)b_{k+1} &= (k + 1)a_{k+1} - \sum_{i=0}^{k-1} (i + 1)b_{i+1}a_{k-i} \\
b_{k+1} &= a_{k+1} - \frac{1}{k + 1} \sum_{i=0}^{k-1} (i + 1)b_{i+1}a_{k-i}
\end{aligned}$$

当然，$b_0 = 0$。这样就可以 $O(n^2)$ 递推出来 0 到 n 次项了。

所以我们就可以在 $O(n^2 2^n)$ 的时间内求出连通生成子图的个数。

7 总结

集合幂级数给我们打开了一扇门，对于以集合为状态的递推把集合幂级数作为其生成函数，我们可以从整体上来分析问题再用快速算法解决。

有关集合幂级数仍有许多有趣的问题，有待我们进一步研究和探索。愿此文对读者有所启发，创造出更加炫酷的应用。

8 感谢

感谢彭雨翔同学，他的那种学到了一点新科技就出成 OI 题给大家分享的精神令人感动，我从他那里学到了许多。本文也是从他的一篇博客延伸而来的。

感谢上海交通大学的郭晓旭学长，他就像移动信息学百科全书一样，平时帮助了我许多。子集卷积的快速算法我最初是从他那里听说的。

感谢金策和杨定澄同学为本文审稿。

感谢中国计算机学会提供学习和交流的平台。

参考文献

[1] Andreas Björklund and Petteri Kaski and Thore Husfeldt, “Fourier meets Möbius: fast subset convolution”. In: Proceedings of the 39th Annual ACM Symposium on Theory of Computing, pp. 67–74. ACM (2007)

[2] 彭雨翔，“Fast Walsh-Hadamard Transform”，http://picks.logdown.com/posts/179290-fast-walsh-hadamard-transform