# 研究性学习结题报告书

2020 年 10 月 24 日

**课题名称** 用信息方法研究遗传学问题

**课题负责人** 杨景云

**课题成员** blablabla

**指导教师** 李丽华老师

**所在班级** 高二 (9) 班

# 1 约定

真值运算符　　若 [] 内表达式为真，则是 1，否则是 0。

# 2 定义

## 2.1 基因集合

我们用 $\mathbb{G}$ 来表示基因集合。

对于只有显隐性的情况，基因集合由一系列大写字母和小写字母组成，大写字母表示显性，小写字母表示隐性。对于只有两对等位基因 A,B 的情况，$\mathbb{G}=\{\mathtt{A},\mathtt{B},\mathtt{a},\mathtt{b}\}$。

对于另一些更复杂的情况，拿喷瓜举例，基因集合可以写作 $\mathbb{G}=\{\mathtt{g}^-,\mathtt{g}^+,\mathtt{G}\}$。

## 2.2 对于集合元素的标号

创建基因集合到 $\{1,2,\cdots|\mathbb{G}|\}$ 的映射 $f:\mathbb{G}\to\mathbb{Z}$。

基因的顺序就是标号的顺序。

容易发现其有逆运算 $f'$。

## 2.3 集合到向量的转化

一个集合 $S$ 可以转化为一个 $|S|$ 维向量 $v$，其中 $v_i=[f'(i)\in S]$。

若基因集合为 {A,B}，A 标号为 1，B 标号为 2，那么集合 {A} 可以转化为 $(1,0)$。

## 2.4 基因片段

基因片段是一个向量。记基因片段组成的集合为 $\mathbb{P}$。

### 配子基因片段

我们用 $\vec{G}$ 来表示配子基因片段。

我们可以将一个具有 $k$ 个基因的配子用一个 $k$ 维向量 $\{a_i\}$ 表示，$a_i\in\mathbb{G}$。

### 个体基因片段

我们用 $\vec{I}$ 来表示个体基因片段。

我们可以将一个具有 $k$ 对等位基因的个体用一个 $k$ 维向量 $\{(l_i,r_i)\}$ 表示，$l_i,r_i\in\mathbb{G}$。

## 2.5 基因片段的运算

### 2.5.1 加法运算 +

对于 $L,R\in\mathbb{P}$，而且 $L,R$ 同为配子基因片段或个体基因片段，定义加法运算为两基因片段的有序拼接。

如 $(\mathtt{A},\mathtt{C})+(\mathtt{B})=(\mathtt{A},\mathtt{B},\mathtt{C})$。

#### 2.5.2 结合运算 $\oplus$

对于 $L, R \in \mathbb{P}$，而且 $L, R$ 同为配子基因片段，而且长度相等，定义结合运算为按位有序结合：

$$(L \oplus R)_i = (\max(L_i, R_i), \min(L_i, R_i))$$

$\max, \min$ 为取序号较大/较小者。
如 $(\mathtt{A}, \mathtt{b}) + (\mathtt{a}, \mathtt{B}) = ((\mathtt{A}, \mathtt{a}), (\mathtt{B}, \mathtt{b}))$。

### 2.6 生成函数 (Generating function)

定义：

$$A = \sum_i a_i x^i$$

是序列 $\{a_i\}$ 的生成函数。
我们不关心 $x$ 的取值和级数是否收敛，把 $x$ 作为形式，只关心系数 $a_i$。

### 2.7 基因片段生成函数

定义：

$$A = \sum_{i \in \mathbb{P}} a_i x^i$$

是序列 $\{a_i\}$ 的基因片段生成函数。

### 2.8 基因片段生成函数的系列运算

**乘法运算 $\times$**

$$x^L \times x^R = x^{L+R}$$

**结合乘法运算 $\otimes$**

$$x^L \otimes x^R = x^{L \oplus R}$$

### 2.9 基因片段生成函数的应用

**求基因型为 AaBB 的个体产生的配子数量比**
构造生成函数：

$$\begin{aligned} G &= (\frac{1}{2}x^{\mathtt{A}} + \frac{1}{2}x^{\mathtt{a}})(\frac{1}{2}x^{\mathtt{B}} + \frac{1}{2}x^{\mathtt{B}}) \\ &= \frac{1}{2}x^{\mathtt{AB}} + \frac{1}{2}x^{\mathtt{aB}} \end{aligned}$$

即配子数量比为 $\texttt{AB} : \texttt{aB} = 1 : 1$。

**求其自交后个体的基因型比例**

构造生成函数：

$$\begin{aligned} I &= G \otimes G \\ &= \frac{1}{4}x^{\texttt{AABB}} + \frac{1}{2}x^{\texttt{AaBB}} + \frac{1}{4}x^{\texttt{aaBB}} \end{aligned}$$

即基因型数量比为 $\texttt{AABB} : \texttt{AaBB} : \texttt{aaBB} = 1 : 2 : 1$。

## 2.10 表现型集合

定义 $\mathbb{E}$ 为表现型集合，一般地，$\mathbb{E} = \mathbb{G}$。

## 2.11 表现型映射

我们创建映射：$\exp : \mathbb{G} \times \mathbb{G} \to \mathbb{E}$，对于一对等位基因 $l, r \in G$ 使得 $\exp(l, r)$ 为这个个体的表现型。

比如 $\exp(\texttt{A}, \texttt{a}) = \texttt{A}$，$\exp(\texttt{a}, \texttt{a}) = \texttt{a}$。

## 2.12 表现型映射的性质

- $\exp(i, j) = \exp(j, i)$。
- $\exp(i, i) = i$。

## 2.13 计算个体的表现型

个体的表现型可以用一个 $k$ 维向量 $\vec{E}$ 表示，其中

$$\vec{E_i} = \exp(\vec{I_i})$$

## 2.14 卷积

给定环 $R$ 上的 $n$ 维向量 $\vec{A} = \{a_i\}, \vec{B} = \{b_i\}$ 和下标运算 $\circ$，设 $C = \{c_i\} = A * B$，则满足：

$$c_i = \sum_{j,k} [j \circ k = i] a_j b_k \tag{1}$$

称 $C$ 为 $A$ 和 $B$ 关于 $\circ$ 的离散卷积，以下简称卷积。

记 $C = A *_{\circ} B$，如果不引起混淆，简记为 $C = A * B$，其中 $*$ 为卷积算子。

若 $\circ = +$，就是我们熟悉的多项式乘法运算。

## 2.15 卷积与生成函数运算的联系

若满足运算 $x^L \times x^R = x^{L \circ R}$，那么生成函数 $F = \sum f_i x^i$ 的乘法：

$$H = F \times G$$

和卷积 $\vec{F}=\{f_i\},\vec{G}=\{g_i\},\vec{H}=\vec{F}*_\circ\vec{G}=\{h_i\}$ 等价。

# 3 只有显隐性情况群体自由交配的计算

参考 2.9 中做法，我们分步计算。

1. 对于第 $i$ 个个体，求配子生成函数 $G_i$。
2. 计算 $G=\sum_{i=1}^{n}G_i$。
3. 计算 $I=G\otimes G$，

## 3.1 配子生成函数的求法

将基因片段对应到一个二进制数，如 $\texttt{AB}=(11)_2=3,\texttt{aB}=(01)_2=1$。

### 朴素求法

模拟生成配子的过程，每次生成一个长度为 $k$ 的二进制数，若第 $i$ 位为 0，则选择第 $i$ 对等位基因的其中一个，否则选择另一个。

拿 `AaBB` 举例：

表 1: 配子计算表

| 选择的二进制数 | 得到的配子 |
|---|---|
| 00 | AB |
| 01 | AB |
| 10 | aB |
| 11 | aB |

生成二进制数的时间复杂度（time complexity）为 $\mathcal{O}(2^k)$，而计算配子的时间复杂度为 $\mathcal{O}(k)$。

所以总时间复杂度是 $\mathcal{O}(k2^k)$，对于 $n$ 个个体都计算一次，时间复杂度为 $\mathcal{O}(nk2^k)$，是不能接受的。

### 快速做法

考虑维护配子出现次数函数 $f$，一开始为 $x^{\texttt{None}}$，考虑每次加入一对基因，$f$ 的变化。假设它变为 $f'$。

若加入的基因是一对显性基因，如 `AA`，那么 $f'(x\times 2+1)=2f(x)$。

若加入的基因是一个显性和一个隐形基因，如 `Aa`，那么 $f'(x\times 2+1)=f(x),f'(x\times 2)=f(x)$。

若加入的基因是一对隐性基因，如 `aa`，那么 $f'(x\times 2)=2f(x)$。

加入 $k$ 等位基因，每次都 $\mathcal{O}(2^k)$ 计算，时间复杂度和上面没有区别，看似没有优化。

但是程序处理时,加入到第 $i$ 个等位基因时,可以只用考虑 $0\sim 2^i$ 的函数值,总时间复杂度是 $\mathcal{O}(\sum_{i=1}^{k}2^i)=\mathcal{O}(2^k)$，可以将一个 $k$ 优化掉。

对于 $n$ 个个体都计算一次，时间复杂度为 $\mathcal{O}(n2^k)$，比较快速。

## 3.2 基因片段生成函数的求法

我们想求出一个基因片段生成函数乘法的快速实现。

### 朴素做法

考虑朴素地实现 (1) 中的卷积，时间复杂度是 $\mathcal{O}(4^k)$，是不能接受的。

### 优化的第一步

我们发现 **对于只有显隐性情况的基因片段生成函数，可以转化为集合生成函数**。而且集合生成函数已经存在快速算法。

### 集合生成函数

可以使用符号：

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

来表示一个集合生成函数。

这里我们定义算子 $\circ = \cup$，即：$x^L \times x^R = x^{L \cup R}$。

容易发现集合生成函数的乘法运算恰好为集合并卷积。

### 基因片段生成函数到集合生成函数的转换

定义全集 $U$ 是：$\{\mathtt{A}, \mathtt{B}, \cdots\}$。

我们将基因片段中的显性基因抽取出来，形成一个集合，如 $\mathtt{ABc} \Rightarrow \{\mathtt{A}, \mathtt{B}\}$。

这样发现集合并卷积刚好符合“显性基因克制隐形基因”的条件，因为只要某一位有对应的显性基因，那么个体就表现为显性，可以结合集合运算表来理解：

表 2: 集合运算表

| $\cup$ | $\{\mathtt{A}\}$ | $\varnothing$ |
|---|---|---|
| $\{\mathtt{A}\}$ | $\{\mathtt{A}\}$ | $\{\mathtt{A}\}$ |
| $\varnothing$ | $\{\mathtt{A}\}$ | $\varnothing$ |

### 集合生成函数的快速卷积算法：FWT

仿照 FFT 的思路，**我们求出 $f$ 的一种变换 $\hat{f}$，使得 $f * g = h \Rightarrow \hat{f}_i \times \hat{g}_i = \hat{h}_i$**，即将系数表示法转化为点值表示法。

我们给出关于集合并卷积的 FWT 运算，即快速莫比乌斯变换。

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

证明：

$$\begin{aligned}\hat{h}_S &= \sum_L \sum_R [(L \cup R) \subseteq S] f_L g_R \\ &= \sum_L \sum_R [L \subseteq S][R \subseteq S] f_L g_R \\ &= \sum_L [L \subseteq S] f_L \sum_R [R \subseteq S] g_R \\ &= \hat{f}_S \hat{g}_S\end{aligned}$$

我们求出 $\hat{h}_S$ 后，当然需要将 $\hat{h}$ 转化为 $h$，于是需要反演运算：

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

可以用容斥简单证明。

### 朴素的变换和反演的实现

枚举 $T$ 和 $S$，并且判断是否 $T \subseteq S$，时间复杂度 $\mathcal{O}(4^k)$，没有太大的变化。

### 经过优化的变换和反演的实现

通过程序精细实现，能够以 $\mathcal{O}(2^{|S|})$ 的时间复杂度枚举 $S$ 的子集。
如果对于所有的 $S \subseteq U$，都这样枚举子集 $T$，时间复杂度为：

$$\mathcal{O}(\sum_{i=0}^{k} \binom{k}{i} 2^i) = \mathcal{O}(3^k)$$

比上述做法稍有进步。

### 进一步优化的变换和反演的实现

我们使用递推的思路，推导出 $\hat{f}_S$。
设 $\hat{f}_S^{(i)} = \sum_{T \subseteq S} [(S \setminus T) \subseteq \{1, \cdots, i\}] f_T$，$\hat{f}_S^{(n)}$ 即是目标序列。
首先有 $\hat{f}_S^{(0)} = f_S$，因为只有当 $S \setminus T$ 为空集时，才能属于空集。
对于所有 $i \notin S$ 的 $S$，满足 $\hat{f}_S^{(i)} = \hat{f}_S^{(i-1)}, \hat{f}_{S \cup \{i\}}^{(i)} = \hat{f}_S^{(i-1)} + \hat{f}_{S \cup \{i\}}^{(i-1)}$。
我们解释一下两个式子。

$$\begin{aligned}\hat{f}_S^{(i)} &= \sum_{T \subseteq S} [(S \setminus T) \subseteq \{1, \cdots, i\}] f_T \\ &= \sum_{T \subseteq S} [(S \setminus T) \subseteq \{1, \cdots, i-1\}] f_T \\ &= \hat{f}_S^{(i-1)}\end{aligned}$$

这里我们发现 $i \notin (S \setminus T)$，所以可以直接把 $\{i\}$ 去掉，也是等价的。

$$
\begin{aligned}
\hat{f}^{(i)}_{S\cup\{i\}} &= \sum_{T\subseteq(S\cup\{i\})} [(S\cup\{i\})\setminus T)\subseteq\{1,\cdots,i\}]f_T \\
&= \sum_{T\subseteq(S\cup\{i\}) \text{ and } i\notin T} [((S\cup\{i\})\setminus T)\subseteq\{1,\cdots,i\}]f_T + \sum_{T\subseteq(S\cup\{i\}) \text{ and } i\in T} [((S\cup\{i\})\setminus T)\subseteq\{1,\cdots,i-1\}]f_T \\
&= \sum_{T\subseteq S \text{ and } i\notin T} [(S\setminus T)\subseteq\{1,\cdots,i-1\}]f_T + \sum_{T\subseteq(S\cup\{i\}) \text{ and } i\in T} [((S\cup\{i\})\setminus T)\subseteq\{1,\cdots,i-1\}]f_T \\
&= \hat{f}^{(i-1)}_S + \hat{f}^{(i-1)}_{S\cup\{i\}}
\end{aligned}
$$

这样，我们 $\mathcal{O}(n2^n)$ 求出 $\hat{f}_S, \hat{g}_S$，按位乘，然后再反演回去即可。

**快速莫比乌斯变换和反演的伪代码实现**

**算法 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

```
function FastMobiusTransform(f)
    for i ← 1 to n do
        for all S ⊆ U \ {i} do
            f_{S∪{i}} ← f_{S∪{i}} + f_S
        end for
    end for
    return f
end function
```

**算法 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

```
function FastMobiusInversion(f)
    for i ← 1 to n do
        for all S ⊆ U \ {i} do
            f_{S∪{i}} ← f_{S∪{i}} − f_S
        end for
    end for
    return f
end function
```

# 4 只有显隐性情况群体自由交配的计算的推广

## 4.1 共显性问题

有一种花卉，基因型为 AA 时表现为红色，基因型为 Aa 时表现为粉色，基因型为 aa 时表现为白色。

表 3: 共显性表现型表

| exp | A | a |
|---|---|---|
| A | A | Aa |
| a | Aa | a |

我们将基因片段中的显性和隐性基因抽取出来，形成一个集合，如 $\mathtt{ABc} \Rightarrow \{\mathtt{A},\mathtt{B},\mathtt{c}\}$，对这样的集合作集合并卷积，也可以理解为把一对等位基因拆成两位，$\mathtt{A} \Rightarrow 10$，$\mathtt{a} \Rightarrow 01$。

容易发现这样做的时间复杂度为 $\mathcal{O}(2k \times 2^{2k}) = \mathcal{O}(2k \times 4^k)$，和朴素做法差不多，是不可接受的。

## 4.2 喷瓜问题

喷瓜的性别由等位基因 $\mathtt{g}^-, \mathtt{g}^+, \mathtt{G}$ 决定，其中：

表 4: 喷瓜表现型表

| exp | $\mathtt{g}^-$ | $\mathtt{g}^+$ | G |
|---|---|---|---|
| $\mathtt{g}^-$ | $\mathtt{g}^-$ | $\mathtt{g}^+$ | G |
| $\mathtt{g}^+$ | $\mathtt{g}^+$ | $\mathtt{g}^+$ | G |
| G | G | G | G |

容易发现，这些等位基因构成一个*偏序集*，我们发现若 $\mathtt{g}^- \leq \mathtt{g}^+ \leq \mathtt{G}$，则 exp 运算对应 max 运算。

将 $\mathtt{g}^-, \mathtt{g}^+, \mathtt{G}$ 编码成为 $00, 01, 10$，那么容易看出：

表 5: 编码运算表

| or | $00(\mathtt{g}^-)$ | $01(\mathtt{g}^+)$ | $10(\mathtt{G})$ |
|---|---|---|---|
| $00(\mathtt{g}^-)$ | $00(\mathtt{g}^-)$ | $01(\mathtt{g}^+)$ | $10(\mathtt{G})$ |
| $01(\mathtt{g}^+)$ | $01(\mathtt{g}^+)$ | $01(\mathtt{g}^+)$ | $11(\mathtt{G})$ |
| $10(\mathtt{G})$ | $10(\mathtt{G})$ | $11(\mathtt{G})$ | $10(\mathtt{G})$ |

发现 $11, 10$ 都对应 G，而 $00$ 对应 $\mathtt{g}^-$，$01$ 对应 $\mathtt{g}^+$。我们在程序实现时最后一步处理一下即可。

容易发现这样做的时间复杂度还是 $\mathcal{O}(2k \times 2^{2k}) = \mathcal{O}(2k \times 4^k)$，和朴素做法差不多，是不可接受的。

# 5 高维 FWT

以上两个问题在只运用集合并卷积的情况下，都没有较低时间复杂度的算法，下面，我们引入高维 FWT，并且逐渐探寻 FWT 的一般式。

## 5.1 定义

定义生成函数 $F=\sum f_S x^S$，其中 $S$ 不再是一个集合，而是一个每维可以取 $0,\cdots,k-1$ 的向量。

## 5.2 每维取 max 的 FWT

容易看出，当 $k=2$，而且：

$$0\circ 0=0$$
$$1\circ 0=1$$
$$0\circ 1=1$$
$$1\circ 1=1$$

那么，这就对应了集合并卷积。

这里，我们不再讨论集合并卷积，而是考虑更加一般的形式，即 $\circ=\max$ 时的情形。

定义：

$$\hat{f}[x^S]=\sum[S\circ T=S]f_T$$

容易发现：

$$\hat{h}_S=\sum_L\sum_R[S\circ(L\circ R)=S]f_L\times g_R$$

由于：

$$\max(a,\max(b,c))=a\Leftrightarrow\max(a,b)=a\,\text{and}\,\max(a,c)=a$$

有：

$$[(S\circ(L\circ R))=S]=[(S\circ L)=S][(S\circ R)=S]$$

得：

$$\begin{aligned}\hat{h}_S&=\sum_L\sum_R[(S\circ L)=S][(S\circ R)=S]f_L\times g_R\\&=\sum_L[(S\circ L)=S]f_L\times\sum_R[(S\circ R)=S]g_R\\&=\hat{f}_S\times\hat{g}_S\end{aligned}$$

那么，我们在 FWT 的 $k$ 个向量中，取前缀和即可，如果是反演的话，相邻做差即可。

## 5.3 喷瓜问题的快速算法

通过上述算法，将 $\mathtt{g}^-$ 对应到 0，$\mathtt{g}^+$ 对应到 1，$\mathtt{G}$ 对应到 2，我们就可以解决上述的喷瓜问题，时间复杂度为 $\mathcal{O}(n \times 3^n)$。

## 5.4 任意操作符的 FWT 问题

容易发现，每次 FWT，都是在对其他位相同，而某一位分别为 $0, \cdots, k-1$ 的 $k$ 个向量对应的下标做矩阵乘法。

如集合并卷积的矩阵：

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

集合交卷积的矩阵：

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

集合对称差卷积的矩阵：

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

上述 max 卷积的矩阵：

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & \cdots & 0 \\ 1 & 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \cdots & 1 \end{bmatrix}$$

而反演则是乘对应的逆矩阵。

我们设矩阵为：

$$\mathbf{M} = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \cdots & a_{0,k-1} \\ a_{1,0} & a_{1,1} & a_{1,2} & \cdots & a_{1,k-1} \\ a_{2,0} & a_{2,1} & a_{2,2} & \cdots & a_{2,k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{k-1,0} & a_{k-1,1} & a_{k-1,2} & \cdots & a_{k-1,k-1} \end{bmatrix}$$

由于 FWT 按位独立，对于某一维分析，有：

$$\left(\begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{k-1} \end{bmatrix} \times \mathbf{M}\right) \cdot \left(\begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ \vdots \\ g_{k-1} \end{bmatrix} \times \mathbf{M}\right) = \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ \vdots \\ h_{k-1} \end{bmatrix} \times \mathbf{M}$$

其中 $\cdot$ 代表“按位乘”，即：

$$\begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{k-1} \end{bmatrix} \cdot \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ \vdots \\ g_{k-1} \end{bmatrix} = \begin{bmatrix} f_0 \times g_0 \\ f_1 \times g_1 \\ f_2 \times g_2 \\ \vdots \\ f_{k-1} \times g_{k-1} \end{bmatrix}$$

由于 $h_i = \sum_{j,k}[j \circ k = i] f_j \times g_k$，枚举每个 $i$，对于每个 $f_j \times g_k$ 分析，容易列出方程：

$$a_{i,j} \times a_{i,k} = a_{i,j \circ k}$$

发现不管对于哪个 $i$，方程都是一样的，去掉 $i$，我们就只用解方程 $a_j \times a_k = a_{j \circ k}$。
如，当 $\circ$ 运算为取 or 的时候，有：

$$\begin{cases} a_0 \times a_0 = a_0 \\ a_1 \times a_0 = a_1 \\ a_0 \times a_1 = a_1 \\ a_1 \times a_1 = a_1 \end{cases}$$

我们解出两组解：

$$\begin{cases} a_0 = 1 \\ a_1 = 0 \end{cases} \qquad \begin{cases} a_0 = 1 \\ a_1 = 1 \end{cases}$$

于是可以这样安排我们的矩阵：

$$\mathbf{M_1} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

$$\mathbf{M_2} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

为什么不能这样这样安排：

$$\mathbf{M_3} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$$

$$\mathbf{M_4} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

原因是，这两个矩阵都没有对应的逆矩阵，求逆矩阵可以再列出一个方程，然后解出 $x_0, x_1$。（可以注意到求解方程的意义正好对应了 FWT 逆操作的意义）

$$\begin{cases} a_{0,0}x_0 + a_{0,1}x_1 = b_0 \\ a_{1,0}x_0 + a_{1,1}x_1 = b_1 \end{cases}$$

拿 $\mathbf{M_1}$ 举例，有：

$$\begin{cases} x_0 = b_1 \\ x_0 + x_1 = b_2 \end{cases}$$

那么显然：

$$\begin{cases} x_0 = b_1 \\ x_1 = b_2 - b_1 \end{cases}$$

于是其逆矩阵就是：

$$\mathbf{M_1^{-1}} = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}$$

而对于 $\mathbf{M_3}$ 来说，有：

$$\begin{cases} x_0 = b_1 \\ x_0 = b_2 \end{cases}$$

显然不合法。于是不能使用 $\mathbf{M_3}$ 这个矩阵。

这样，我们解出矩阵 $\mathbf{M}$，然后求出逆矩阵 $\mathbf{M^{-1}}$，就可以解决任意操作符 $\circ$ 的 FWT 问题。

### 5.5 $\circ$ 运算需要满足的性质

由于：

$$a_j \times a_k = a_{j \circ k}$$

有：

$$a_{j \circ k} = a_j \times a_k = a_k \times a_j = a_{k \circ j}$$

$$a_{j \circ (k \circ l)} = a_j \times a_{k \circ l} = a_j \times a_k \times a_l = a_{j \circ k} \times a_l = a_{(j \circ k) \circ l}$$

于是 $\circ$ 运算必须满足交换律和结合律。

## 5.6 不进位加法的 FWT

我们定义不进位加法 $\oplus_p$ 运算，为：

$$a \oplus_p b = \begin{cases} a+b & (0 \le a+b \le p-1) \\ a+b-p & (p \le a+b \le 2p-2) \end{cases}$$

容易发现，其矩阵系数 $a$ 满足：

$$a_{i,j} \times a_{i,k} = a_{i,j \oplus_p k}$$

这里，我们发现，这组方程的特解即是：

$$a_{i,j} = \omega_p^j$$

因为单位根运算满足：

$$\omega_p^k = \omega_p^{k+p}$$

$$\omega_p^{i+j} = \omega_p^i \times \omega_p^j$$

进而发现，方程有 $p$ 组解，第 $i$ 组解（从 0 开始编号）为：

$$a_{i,j} = \omega_p^{j \times i}$$

那么我们可以列出矩阵：

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_p^1 & \omega_p^2 & \cdots & \omega_p^{p-1} \\ 1 & \omega_p^2 & \omega_p^4 & \cdots & w_p^{2(p-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_p^{p-1} & \omega_p^{2(p-1)} & \cdots & \omega_p^{(p-1)(p-1)} \end{bmatrix}$$

此矩阵就是范德蒙德矩阵。

我们不加证明地给出它的逆矩阵：

$$\frac{1}{p} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_p^{-1} & \omega_p^{-2} & \cdots & \omega_p^{-(p-1)} \\ 1 & \omega_p^{-2} & \omega_p^{-4} & \cdots & \omega_p^{-2(p-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_p^{-(p-1)} & \omega_p^{-2(p-1)} & \cdots & \omega_p^{-(p-1)(p-1)} \end{bmatrix}$$

这样，我们就可以完成模 $p$ 意义下的不进位加法卷积，此算法即多维广义离散傅里叶变换。

具体程序实现，我们可以算出 $\omega_p^1$ 即 $\cos\frac{p}{2\pi} + i\sin\frac{p}{2\pi}$，如果能用根号形式表示，即：$a+b\sqrt{x}i$，我们可以模拟复数 $a+b\sqrt{x}i$，其乘法为 $(a+b\sqrt{x}i)(c+d\sqrt{x}i) = (ac-xbd)+(ac+bd)\sqrt{x}i$，加法为 $(a+b\sqrt{x}i)+(c+d\sqrt{x}i) = (a+b)+(c+d)\sqrt{x}i$。

或者，更加通用地，我们将长度为 $p$ 的多项式环作为一种数据结构，假设是 $F=\sum_{i=0}^{p-1}\omega_p^i f_i$，有：$F\times\omega_p^k=F=\sum_{i=0}^{p-1}\omega_p^i f_{i\ominus k}$，$F\times G=\sum_{i=0}^{p-1}\sum_{j=0}^{p-1}w_p^{i\oplus j}f_i\times g_j$。

注意，如果对某个数 $mod$ 取模，若 $p$ 在 $mod$ 下没有对应的逆，则不能使用此算法。

多维广义离散傅里叶变换的代码实现：

**算法 3** 多维广义离散傅里叶变换

**输入:** 幂级数 $f$，单位根 $w_p$，操作符 $opr$ 代表正变换还是逆变换。
**输出:** $f$ 的傅里叶变换

```
function FourierTransform(f, w_p, opr)
    if then opr = 1
        M_{i,j} ← w_p^{(i-1)(j-1)}
    else
        M_{i,j} ← (1/p) w_p^{-(i-1)(j-1)}
    end if
    for i ← 1 to n do
        for The p vectors satisfying 1···p on the i-th bit and the other bits are same. do
            v ← the p vectors
            for j ← 1 to p do
                g_j ← f_{v_j}
            end for
            g ← g × M
            for j ← 1 to p do
                f_{v_j} ← g_j
            end for
        end for
    end for
    return f
end function
```

## 5.7 共显性问题的快速算法

如果将 `A` 对应到 1，`a` 对应到 0，容易发现：

表 6: 编码运算表

| + | 1(A) | 0(a) |
|---|---|---|
| 1(A) | 2(A) | 1(Aa) |
| 0(a) | 1(Aa) | 0(a) |

我们发现 `A` 对应 2，`Aa` 对应 1，`a` 对应 0。

只要使用三次单位根 $\omega_3=\cos 120^\circ+\sin 120^\circ\mathrm{i}$，即可轻松解决此问题，时间复杂度是 $\mathcal{O}(n\times 3^n)$。