

集合幂级数

云浅

2025 年 5 月

Content

1 and/or/xor 卷积

- Fast Mobius Transform
- Fast Walsh Transform
- 例题

2 子集卷积

- 集合不交并卷积
- 半半在线卷积
- 集合幂级数的多项式操作

3 题目选讲

and/or/xor 卷积

- 考虑两个长为 2^n 的序列 $A_{0\dots 2^n-1}, B_{0\dots 2^n-1}$
- 它们的 and/or/xor 卷积 $C = A * B$ 定义为

$$C_k = \sum_{i \oplus j = k} = A_i \times B_j$$

- 其中 \times 是数乘, \oplus 是 and/or/xor 中的某种位运算。

and/or/xor 卷积

- 考虑两个长为 2^n 的序列 $A_{0\dots 2^n-1}, B_{0\dots 2^n-1}$
- 它们的 and/or/xor 卷积 $C = A * B$ 定义为

$$C_k = \sum_{i \oplus j = k} = A_i \times B_j$$

- 其中 \times 是数乘, \oplus 是 and/or/xor 中的某种位运算。
- 这三种卷积都可以在 $O(n2^n)$ 的时间内计算完毕。

and/or/xor 卷积

- 考虑两个长为 2^n 的序列 $A_{0\dots 2^n-1}, B_{0\dots 2^n-1}$
- 它们的 and/or/xor 卷积 $C = A * B$ 定义为

$$C_k = \sum_{i \oplus j = k} = A_i \times B_j$$

- 其中 \times 是数乘， \oplus 是 and/or/xor 中的某种位运算。
- 这三种卷积都可以在 $O(n2^n)$ 的时间内计算完毕。
- 注意到 or/and 可以对应上集合的并和交，因此在下文中，我们不再区分集合与二进制数，即认为 i or j 与 $i \cup j$ 相同。

集合幂级数

类似多项式乘法，如果我们记多项式 $F(x) = \sum_{S \subseteq U} f_S x^S$ ，那么 and/or/xor 卷积可以看作这样的多项式的多项式乘法。

我们把这样的多项式叫做集合幂级数。那么 and/or/xor 卷积就是定义 $x^S \times x^T = x^{S \cap T / S \cup T / S \text{ xor } T}$ 时的多项式乘法。

后面我们会引入集合不交并卷积即子集卷积，这其实就是定义

$$x^S \times x^T = \begin{cases} x^{S \cup T} & , S \cap T = \emptyset \\ 0 & , \text{otherwise.} \end{cases}$$

Fast Möbius Transform

我们先来解决 or 卷积。显然可以直接枚举子集做到 $O(3^n)$ 。

Fast Möbius Transform

我们先来解决 or 卷积。显然可以直接枚举子集做到 $O(3^n)$ 。

发现 or 相当于在每一位上取 max，不妨考虑一维的 max 卷积：

$$C_k = \sum_{\max(i,j)=k} A_i \times B_j$$

Fast Möbius Transform

我们先来解决 or 卷积。显然可以直接枚举子集做到 $O(3^n)$ 。

发现 or 相当于在每一位上取 max，不妨考虑一维的 max 卷积：

$$C_k = \sum_{\max(i,j)=k} A_i \times B_j$$

考虑容斥，先计算 $\sum_{\max(i,j) \leq k} A_i \times B_j$ ，这样只需要再做一次差分就可以得到 C 了。

Fast Möbius Transform

注意到 $[\max(i, j) \leq k] = [i \leq k] \times [j \leq k]$, 因此

$$\sum_{\max(i, j) \leq k} A_i \times B_j = \left(\sum_{i \leq k} A_i \right) \times \left(\sum_{i \leq k} B_i \right)$$

我们可以将 A, B 先分别做前缀和, 再做点乘 (即对应位置相乘), 再做差分即可得到 C 。

Fast Möbius Transform

考虑高维的情况。

Fast Möbius Transform

考虑高维的情况。

类似地，我们将 $i \cup j = k$ 改为 $i \cup j \subseteq k$ ，然后拆成

$$[i \subseteq k] \times [j \subseteq k]$$

Fast Möbius Transform

考虑高维的情况。

类似地，我们将 $i \cup j = k$ 改为 $i \cup j \subseteq k$ ，然后拆成

$$[i \subseteq k] \times [j \subseteq k]$$

那么就有

$$\sum_{i \cup j \subseteq k} A_i \times B_j = \left(\sum_{i \subseteq k} A_i \right) \times \left(\sum_{j \subseteq k} B_j \right)$$

Fast Möbius Transform

考虑高维的情况。

类似地，我们将 $i \cup j = k$ 改为 $i \cup j \subseteq k$ ，然后拆成

$$[i \subseteq k] \times [j \subseteq k]$$

那么就有

$$\sum_{i \cup j \subseteq k} A_i \times B_j = \left(\sum_{i \subseteq k} A_i \right) \times \left(\sum_{j \subseteq k} B_j \right)$$

因此，我们将 A, B 先分别做高维前缀和，再做点乘，再做高维差分即可得到 C 。

高维前缀和 & 差分

那么现在只需要计算高维前缀和。

高维前缀和 & 差分

那么现在只需要计算高维前缀和。

如果暴力计算，复杂度仍然是 $O(3^n)$ 。

高维前缀和 & 差分

那么现在只需要计算高维前缀和。

如果暴力计算，复杂度仍然是 $O(3^n)$ 。

考虑二维前缀和的公式：

$$s_{i,j} = a_{i,j} + s_{i-1,j} + s_{i,j-1} - s_{i-1,j-1}$$

高维前缀和 & 差分

那么现在只需要计算高维前缀和。

如果暴力计算，复杂度仍然是 $O(3^n)$ 。

考虑二维前缀和的公式：

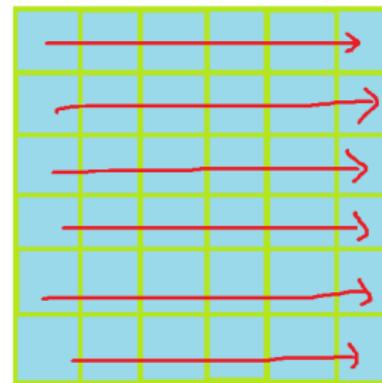
$$s_{i,j} = a_{i,j} + s_{i-1,j} + s_{i,j-1} - s_{i-1,j-1}$$

其原理是容斥，但并不具有很好的推广性：如果要计算 D 维前缀和，那么复杂度至少为 $2^D \times$ 状态数。

高维前缀和 & 差分

考虑二维前缀和的另一种做法：先对第一维做前缀和，再对第二维做前缀和。

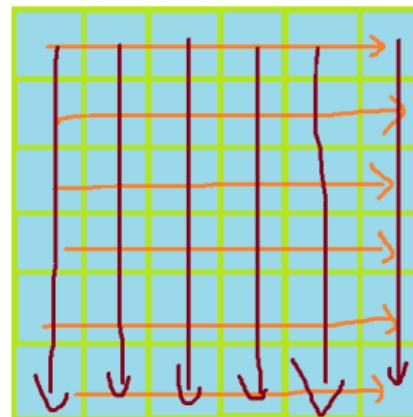
如图所示，我们先对第一维做前缀和：



此时 $s_{i,j}$ 的值是 $\sum_{k \leq j} a_{i,k}$ ，即同一行中他左侧的位置。

高维前缀和 & 差分

再对第二维做前缀和：



现在 $s_{i,j}$ 的值就是正确的 $\sum_{x \leq i, y \leq j} a_{x,y}$ 了。

高维前缀和 & 差分

一个简单的代码：

```
1 for(int i=1;i<=n;i++)
2     for(int j=1;j<=m;j++)add(A[i][j],A[i-1][j]);
3 for(int i=1;i<=n;i++)
4     for(int j=1;j<=m;j++)add(A[i][j],A[i][j-1]);
```

高维前缀和 & 差分

一个简单的代码：

```
1 for(int i=1;i<=n;i++)  
2     for(int j=1;j<=m;j++)add(A[i][j],A[i-1][j]);  
3 for(int i=1;i<=n;i++)  
4     for(int j=1;j<=m;j++)add(A[i][j],A[i][j-1]);
```

这个做法有很好的扩展性，可以简单扩展到 n 维：

```
1 for(int i=0;i<n;i++)  
2     for(int S=0;S<(1<<n);S++)  
3         if(S&(1<<i))add(A[S],A[S^(1<<i)]);
```

时间复杂度为 $O(n2^n)$ 。

集合幂级数

└ and/or/xor 卷积

└ Fast Möbius Transform

高维前缀和 & 差分

高维差分怎么办？

高维前缀和 & 差分

高维差分怎么办？

把刚才的代码反过来写一遍就行了。

```
1 for(int i=n-1;i>=0;i--)  
2     for(int S=0;S<(1<<n);S++)  
3         if(S&(1<<i))add(A[S],mod-A[S^(1<<i)]);
```

高维前缀和 & 差分

高维差分怎么办？

把刚才的代码反过来写一遍就行了。

```
1 for(int i=n-1;i>=0;i--)  
2     for(int S=0;S<(1<<n);S++)  
3         if(S&(1<<i))add(A[S],mod-A[S^(1<<i)]);
```

显然每一位是等价的，因此第一重循环完全可以改成正序的。

高维前缀和 & 差分

高维差分怎么办？

把刚才的代码反过来写一遍就行了。

```
1 for(int i=n-1;i>=0;i--)  
2     for(int S=0;S<(1<<n);S++)  
3         if(S&(1<<i))add(A[S],mod-A[S^(1<<i)]);
```

显然每一位是等价的，因此第一重循环完全可以改成正序的。

实际上，上述过程就是「快速莫比乌斯变换」，即 FMT (Fast Möbius Transform)。

Fast Möbius Transform

因此，FMT 其实就是高维前缀和。代码很好写：

```
1 void FMT(vector<int>&A, int k, int tag){  
2     for(int i=0;i<k;i++) for(int S=0;S<(1<<k);S++)  
3         if(S&(1<<i))  
4             add(A[S], tag==1?A[S^(1<<i)]:mod-A[S^(1<<i)]);  
5 }
```

Fast Möbius Transform

因此，FMT 其实就是高维前缀和。代码很好写：

```
1 void FMT(vector<int>&A, int k, int tag){  
2     for(int i=0;i<k;i++) for(int S=0;S<(1<<k);S++)  
3         if(S&(1<<i))  
4             add(A[S], tag==1?A[S^(1<<i)]:mod-A[S^(1<<i)]);  
5 }
```

那么如果要求两个序列 A, B 的 or 卷积，只需要先将 A, B 做 FMT，将 C 置为 A, B 的点乘，再执行 $\text{FMT}(C, n, -1)$ 即可。

如果需要求 and 卷积，只需要将高维前缀和改成高维后缀和。

Fast Walsh Transform

xor 卷积怎么办？

Fast Walsh Transform

xor 卷积怎么办？

似乎没法简单拆开 xor 的式子。

Fast Walsh Transform

xor 卷积怎么办？

似乎没法简单拆开 xor 的式子。

我们发现 FMT 实际上本质上是将两个向量 A, B 同时乘上一个矩阵 T , 再做点乘，然后乘上 T^{-1} 。即

$$(A \times T) \cdot (B \times T) = (A \star B) \times T$$

其中 \star 表示 or/and 卷积， \cdot 表示向量点乘， \times 表示矩阵乘法。

Fast Walsh Transform

xor 卷积怎么办？

似乎没法简单拆开 xor 的式子。

我们发现 FMT 实际上本质上是将两个向量 A, B 同时乘上一个矩阵 T , 再做点乘，然后乘上 T^{-1} 。即

$$(A \times T) \cdot (B \times T) = (A \star B) \times T$$

其中 \star 表示 or/and 卷积， \cdot 表示向量点乘， \times 表示矩阵乘法。

那么矩阵 T 只需要满足 $T(i, k)T(j, k) = T(i \oplus j, k)$, 且 T 可逆。

在 FMT 中，矩阵 T 就是 $T(i, j) = [i \subseteq j]$ (或者 $[j \subseteq i]$ ，如果要做 and 卷积)。

Fast Walsh Transform

位运算在每一位上是独立的。

Fast Walsh Transform

位运算在每一位上是独立的。

如果我们能找到大小为 2×2 的矩阵 T_0 , 且满足对 $i, j, k \in \{0, 1\}$ 有 $T_0(i, k)T_0(j, k) = T_0(i \text{ xor } j, k)$, 那么如果要对长为 2^n 的序列做变换, 只需要令

$$T(i, j) = \prod_{x=0}^{n-1} T_0(i_x, j_x)$$

其中 i_x 表示 i 在二进制下的第 x 位。

Fast Walsh Transform

这样就有

$$\begin{aligned} T(i, k)T(j, k) &= \prod_{i=0}^{n-1} T_0(i_x, k_x)T_0(j_x, k_x) \\ &= \prod_{i=0}^{n-1} T_0(i_x \text{ xor } j_x, k_x) = T(i \text{ xor } j, k) \end{aligned}$$

Fast Walsh Transform

这样就有

$$\begin{aligned} T(i, k)T(j, k) &= \prod_{i=0}^{n-1} T_0(i_x, k_x)T_0(j_x, k_x) \\ &= \prod_{i=0}^{n-1} T_0(i_x \text{ xor } j_x, k_x) = T(i \text{ xor } j, k) \end{aligned}$$

考虑解一下合法的 2×2 的矩阵 T , 需要

$$T_{0,k}^2 = T_{1,k}^2 = T_{0,k}, T_{0,k}T_{1,k} = T_{1,k}.$$

如果限制在 \mathbb{R} 上, 就需要有 $T_{0,k} = 0$ 或 1 ; 然而 $T_{0,k} = 0$ 会导致 $T_{1,k} = 0$, 矩阵不可逆, 因此必须 $T_{0,k} = 1$ 。接下来有 $T_{1,k} = \pm 1$, 为了让矩阵可逆, 必须一行放一个 1 , 一行放一个 -1 。

Fast Walsh Transform

于是，我们得到了一个合法的 T_0 :

$$T_0 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, T_0^{-1} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} \end{bmatrix}$$

于是也就有 FWT 的那个式子: $FWT(A)[i] = \sum_{j=0}^{2^n-1} (-1)^{|i \cap j|} A_j$ 。

Fast Walsh Transform

现在还需要在低于 $O(2^{2n})$ 时间内计算某个向量乘上矩阵 T 。
仍然是注意到位运算对每一位独立，对每一位分别操作：

Fast Walsh Transform

现在还需要在低于 $O(2^{2n})$ 时间内计算某个向量乘上矩阵 T 。

仍然是注意到位运算对每一位独立，对每一位分别操作：

```
1 for(int i=0;i<k;i++)for(int S=0;S<(1<<k);S++){
2     if(!(S&(1<<i)))continue;
3     int x=A[S^(1<<i)],y=A[S];
4     Mod(A[S^(1<<i)]=x+y),Mod(A[S]=x+mod-y);
5 }
```

时间复杂度 $O(n2^n)$ 。

Fast Walsh Transform

现在还需要在低于 $O(2^{2n})$ 时间内计算某个向量乘上矩阵 T 。
仍然是注意到位运算对每一位独立，对每一位分别操作：

```
1 for(int i=0;i<k;i++)for(int S=0;S<(1<<k);S++){
2     if(!(S&(1<<i)))continue;
3     int x=A[S^(1<<i)],y=A[S];
4     Mod(A[S^(1<<i)]=x+y),Mod(A[S]=x+mod-y);
5 }
```

时间复杂度 $O(n2^n)$ 。

如果要乘上 T^{-1} ，只需要最后把每个数都除掉 2^n 即可。

一般进制表下的 FWT

如果我们给出一个一般的 $k \times k$ 的进制运算表 A , $A(x, y) = z$ 代表 x, y 进行 k 进制下的这个运算会得到 z , 再以此定义两个 k 进制数 x, y 的运算 $x \oplus y$ 为在每一位上进行这样的运算, 并定义新的卷积

$$(A \star B)_z = \sum_{x \oplus y = z} A_x B_y$$

可以证明在二进制下的所有进制表都能归结为 or 和 xor 的运算; 但在更高的进制下并非如此。有兴趣的同学可以结合 [CF102129A](#) 与 [P8970](#) 进行学习。

update: 2025 年的集训队互测中也有一道: QOJ9562

CF449D. Jzzhu and Numbers

给定长为 n 的序列 a , 求有多少个子序列 $1 \leq i_1 < i_2 < \dots < i_k \leq n$
满足 $a_{i_1}, a_{i_2}, \dots, a_{i_k}$ 的按位与为 0。

$1 \leq n \leq 10^6, 0 \leq a_i \leq 10^6$ 。

Jzzhu and Numbers

考虑容斥，按位与为 0 相当于钦定每一位上至少一个 0，容斥成钦定 S 以内的位都必须全是 1，做高维后缀和即可。容斥系数即为 $(-1)^{|S|}$ 。
复杂度是 $O(n + a_i \log a_i)$ 。

CF850E. Random Election

简化版题意：

给定序列 $F[0 \dots 2^n - 1]$, 现在随机选出 n 个长度为 3 的排列 $P_{0 \dots n-1}$ 。
考虑三个正整数 A, B, C , 对每个 $i = 0 \dots n - 1$:

- 若 $P_{i,1} > P_{i,2}$, 则在 A 的二进制中第 i 位写下 1, 否则写下 0。
- 若 $P_{i,2} > P_{i,3}$, 则在 B 的二进制中第 i 位写下 1, 否则写下 0。
- 若 $P_{i,3} > P_{i,1}$, 则在 C 的二进制中第 i 位写下 1, 否则写下 0。

问所有 6^n 种方案中, 最终 $F[A] = F[B] = F[C]$ 的方案数。

Subtask 1: $1 \leq n \leq 16$, 你需要分别算出 $F[A] = F[B] = F[C] = 0$ 与 $F[A] = F[B] = F[C] = 1$ 的方案数。

Subtask 2: $1 \leq n \leq 20$, 你只需算出二者之和。

Random Election

选出的 A, B, C 只需要满足每一位上不能是三个 0 或三个 1。

问题转化为：问有多少个 $0 \leq A, B, C \leq 2^n - 1$ 使得

- A and B and $C = 0$
- A or B or $C = 2^n - 1$
- $F[A] = F[B] = F[C]$ 。

Subtask 1

考虑把 $F[x] = 1$ 的 x 单拎出来。

相当于要从集合 $\{x | F[x] = 1\}$ 中选三个数 A, B, C , 使得一位上至少有一个 0 和一个 1。

Subtask 1

考虑把 $F[x] = 1$ 的 x 单拎出来。

相当于要从集合 $\{x | F[x] = 1\}$ 中选三个数 A, B, C , 使得一位上至少有一个 0 和一个 1。

考虑容斥, 欽定 S 中的位必须全都是 A, B, C 不能有 1。

再对符合第一个条件的这些数求出每一位上至少有一个 0 的方案数, 这就是前面的例题[Jzzhu and Numbers](#), 可以在 $O((n - |S|)2^{n-|S|})$ 时间内解决。

Subtask 1

清算一下复杂度：

$$\sum_S (n - |S|)2^{n-|S|} = \sum_{i=0}^n \binom{n}{i} (n-i)2^{n-i} = 2n \cdot 3^{n-1}$$

故总时间复杂度 $O(n3^n)$ 。

Subtask 2

$F[A] = F[B] = F[C]$ 不好限制，考虑先计算它们不全相等的方案数。
发现只要随便两个不同就行。

Subtask 2

$F[A] = F[B] = F[C]$ 不好限制，考虑先计算它们不全相等的方案数。
发现只要随便两个不同就行。

不妨钦定 $F[A] = 1, F[B] = 0$ ，那么不全相等的方案数就是这个方案数乘上 $6 (= 2 \times 3)$ 。

Subtask 2

$F[A] = F[B] = F[C]$ 不好限制，考虑先计算它们不全相等的方案数。
发现只要随便两个不同就行。

不妨钦定 $F[A] = 1, F[B] = 0$ ，那么不全相等的方案数就是这个方案数乘上 $6 (= 2 \times 3)$ 。

现在相当于从 $F[x] = 1$ 的 x 中选出一个当 A , $F[x] = 0$ 的 x 中选出一个当 B, C 只需要满足每一位上不能是三个 1 或三个 0。

Subtask 2

$F[A] = F[B] = F[C]$ 不好限制，考虑先计算它们不全相等的方案数。发现只要随便两个不同就行。

不妨钦定 $F[A] = 1, F[B] = 0$ ，那么不全相等的方案数就是这个方案数乘上 $6 (= 2 \times 3)$ 。

现在相当于从 $F[x] = 1$ 的 x 中选出一个当 A , $F[x] = 0$ 的 x 中选出一个当 B, C 只需要满足每一位上不能是三个 1 或三个 0。

发现对于每一位，如果 A, B 这一位上相等，那么 C 在这一位上只有一种方案；否则有两种方案。因此方案数就是 $2^{\text{popcount}(x \text{ xor } y)}$ 。

FWT 即可，复杂度 $O(n2^n)$ 。

CF662C. Binary Table

有一个 n 行 m 列的表格，每个元素都是 0/1，每次操作可以选择一行或一列，把 0/1 翻转，即把 0 换为 1，把 1 换为 0。

请问经过若干次操作后，表格中最少有多少个 1。

$1 \leq n \leq 20, 1 \leq m \leq 10^5$ 。

Binary Table

发现如果确定了 n 行的翻转方案，那么 m 列的方案肯定是每行贪心地选。形式化地讲，答案可以表示为

$$\min_S \sum_{i=1}^m \min(|T_i \text{ xor } S|, n - |T_i \text{ xor } S|)$$

其中 T_i 表示第 i 列的二进制数， $|x|$ 表示二进制数 x 的 `popcount`。

Binary Table

考虑枚举 $T_i \text{ xor } S$ 。

设 $g(A) = \sum_{i=1}^m [T_i = A]$ 表示 A 在这 m 列中的出现次数，同时记 $w(A) = \min(|A|, n - |A|)$ ，那么

$$\sum_{i=1}^m w(T_i \text{ xor } S) = \sum_C w(C)g(S \text{ xor } C)$$

现在只需要对每个 S 计算 $\sum_C w(C)g(S \text{ xor } C)$ 。

Binary Table

考虑枚举 $T_i \text{ xor } S$ 。

设 $g(A) = \sum_{i=1}^m [T_i = A]$ 表示 A 在这 m 列中的出现次数，同时记 $w(A) = \min(|A|, n - |A|)$ ，那么

$$\sum_{i=1}^m w(T_i \text{ xor } S) = \sum_C w(C)g(S \text{ xor } C)$$

现在只需要对每个 S 计算 $\sum_C w(C)g(S \text{ xor } C)$ 。

注意到这就是 $\sum_{A \text{ xor } B=S} w(A)g(B)$ ，FWT 即可。复杂度 $O(n2^n)$ 。

不难发现这个方法对于 w 没有要求，因此可以解决许多形如「对每个 S 算 $\sum w(T_i \text{ xor } S)$ 」的问题。

UOJ310. 黎明前的巧克力

有 n 个数 $a_{1,\dots,n}$, 你需要选出两个不交的集合 (可空) S_1, S_2 , 使得 S_1 中所有 a_i 的异或和等于 S_2 中所有数的异或和。

求出方案数对 998244353 取模的值。

$1 \leq n \leq 10^6, 0 \leq a_i \leq 10^6$ 。

黎明前的巧克力

考虑枚举最后的 $S_1 \cup S_2$, 发现转化为:

- 选出一个异或和为 0 的子集 T , 权值为 $2^{|T|}$, 求所有方案的权值之和。

黎明前的巧克力

考虑枚举最后的 $S_1 \cup S_2$, 发现转化为:

- 选出一个异或和为 0 的子集 T , 权值为 $2^{|T|}$, 求所有方案的权值之和。

即求 $[x^0] \prod_{i=1}^n (1 + 2x^{a_i})$, 这里认为 $x^S \times x^T = x^{S \text{ xor } T}$ 。

黎明前的巧克力

这个多项式只有两个位置有值，我们尝试手算 FWT。

黎明前的巧克力

这个多项式只有两个位置有值，我们尝试手算 FWT。
发现每个多项式的 FWT 的每一项只可能是 -1 或 3 。

黎明前的巧克力

这个多项式只有两个位置有值，我们尝试手算 FWT。

发现每个多项式的 FWT 的每一项只可能是 -1 或 3 。

考虑直接算出 $\sum \text{FWT}(f_i)$, 然后得到每一位上的和, 如果是 s , 设有 x 个 -1 和 y 个 3 , 那么 $x + y = n, 3y - x = s$, 可以直接解出来

$$y = \frac{n+s}{4}, x = \frac{3n-s}{4}.$$

最后的贡献就是 $(-1)^x 3^y$, 做一遍 IFWT 即可。

黎明前的巧克力

这个多项式只有两个位置有值，我们尝试手算 FWT。

发现每个多项式的 FWT 的每一项只可能是 -1 或 3 。

考虑直接算出 $\sum \text{FWT}(f_i)$, 然后得到每一位上的和, 如果是 s , 设有 x 个 -1 和 y 个 3 , 那么 $x + y = n, 3y - x = s$, 可以直接解出来

$$y = \frac{n+s}{4}, x = \frac{3n-s}{4}.$$

最后的贡献就是 $(-1)^x 3^y$, 做一遍 IFWT 即可。

由于 $\sum \text{FWT}(f_i) = \text{FWT}(\sum f_i)$ 可以 $O(a_i \log a_i)$ 算出, 总的复杂度是 $O(a_i \log a_i + n)$ 。

Jzzhu and Numbers revisit

给定长为 n 的序列 a , 求有多少个子序列 $1 \leq i_1 < i_2 < \dots < i_k \leq n$
满足 $a_{i_1}, a_{i_2}, \dots, a_{i_k}$ 的按位与为 0。

$1 \leq n \leq 10^6, 0 \leq a_i \leq 10^6$ 。

Jzzhu and Numbers revisit

答案即为 $[x^0] \prod (1 + x^{a_i})$, 这里认为 $x^S \times x^T = x^{S \cap T}$ 。

类似上题, 每个 $f_i(x) = 1 + x^{a_i}$ 的 FMT 中只可能有 1 和 2, 可以简单算出 $\sum \text{FMT}(f_i(x))$ 然后解出这一位上 1 和 2 的个数。

这样就得到了 $\prod_{i=1}^n \text{FMT}(f_i(x))$, 做一遍 IFMT 即可。

复杂度 $O(n + a_i \log a_i)$ 。

更一般的形式

上面两道题都是在计算这样的式子：

$$\prod_{i=1}^n \sum_{j=0}^r w_j x^{a_{i,j}}$$

其中认为 $x^S \times x^T = x^{S \text{ xor } T} / x^{S \cap T} / x^{S \cup T}$, 等等。

更一般的形式

上面两道题都是在计算这样的式子：

$$\prod_{i=1}^n \sum_{j=0}^r w_j x^{a_{i,j}}$$

其中认为 $x^S \times x^T = x^{S \text{ xor } T} / x^{S \cap T} / x^{S \cup T}$, 等等。

一般地, 设 a 的值域是 $[0, 2^k)$, 每个多项式中有 r 项, 那么我们可以在 $O(nr2^r + (r+k)2^{r+k})$ 时间内算出这个乘积。

感兴趣的同学可以参考 [CF1119H Triples](#) 进行学习。

Luogu7526. Virtual Self 弱化版

给定序列 $v_{0 \dots 2^w - 1}$, 有 $n = \sum v_i$ 簇粒子, 其中权值为 i 的粒子有 v_i 个。

一个水晶由恰好 m 簇粒子合成, 定义这个水晶的权值为合成他的所有粒子的权值的异或和。

你需要对所有 $k = 0, 1, \dots, 2^w - 1$ 求出, 在所有的 $\binom{n}{m}$ 种水晶中, 有多少个水晶的权值为 k 。对 998244353 取模。

$1 \leq w \leq 20, 0 \leq \sum v_i = n \leq 10^6, 1 \leq m \leq \min(n, 10^6)$ 。

Virtual Self 弱化版

考虑写出 $\sum v_i$ 个多项式，其中 x^i 有 v_i 个，那么答案相当于从这 $\sum v_i$ 个多项式里面选 m 个，求出乘积（其中 $x^a \times x^b = x^{a \text{ xor } b}$ ），再全部求和。

Virtual Self 弱化版

考虑写出 $\sum v_i$ 个多项式，其中 x^i 有 v_i 个，那么答案相当于从这 $\sum v_i$ 个多项式里面选 m 个，求出乘积（其中 $x^a \times x^b = x^{a \text{ xor } b}$ ），再全部求和。

考虑 FWT，相当于选 m 个多项式，把它们 FWT 后乘起来，再对所有这种的 FWT 之和求和，再 IFWT 回去。

Virtual Self 弱化版

考虑写出 $\sum v_i$ 个多项式，其中 x^i 有 v_i 个，那么答案相当于从这 $\sum v_i$ 个多项式里面选 m 个，求出乘积（其中 $x^a \times x^b = x^{a \text{ xor } b}$ ），再全部求和。

考虑 FWT，相当于选 m 个多项式，把它们 FWT 后乘起来，再对所有这种的 FWT 之和求和，再 IFWT 回去。

发现每个序列 FWT 后会有若干位 1 和若干位 -1 ，套路地算出来所有序列的 FWT 之和即可解出这一位上 -1 的个数。假设有 a 个，则答案的 FWT 在这一位上（乘法情形时）的值就是

$$\sum_{i=0}^a (-1)^i \binom{a}{i} \binom{n-a}{m-i}$$

可以拆成卷积形式，用 NTT $O(n \log n)$ 预处理。时间复杂度是 $O(n \log n + w2^w)$ 。

集合不交并卷积

仍然是两个长为 2^n 的序列 $A_{0\dots 2^n-1}, B_{0\dots 2^n-1}$, 它们的集合不交并卷积定义为

$$C_k = \sum_{i \cup j = k, i \cap j = \emptyset} A_i \times B_j$$

集合不交并卷积

仍然是两个长为 2^n 的序列 $A_{0\dots 2^n-1}, B_{0\dots 2^n-1}$, 它们的集合不交并卷积定义为

$$C_k = \sum_{i \cup j = k, i \cap j = \emptyset} A_i \times B_j$$

我们可以在 $O(n^2 2^n)$ 时间内计算两个序列的集合不交并卷积。

占位多项式

考虑构造占位多项式 $w(x) = |x|$, 我们把两个条件拆成 $x \cup y = i$ 与 $|x| + |y| = |i|$ 。

占位多项式

考虑构造占位多项式 $w(x) = |x|$, 我们把两个条件拆成 $x \cup y = i$ 与 $|x| + |y| = |i|$ 。

把所有 $|x| = k$ 的 A_x 分成一组, 记为 f_k (这是一个长为 2^n 的序列, 其中满足 $|x| = k$ 的位置 f_x 的值为 $f_x = A_x$, 其余为 $f_x = 0$), 把 B 那边做相应的分组, 设为 g 。

那么只需要做卷积 $f_i \star g_j \rightarrow h_{i+j}$ (其中 \star 表示两个序列的或卷积), 最后只保留 h_k 中 $|x| = k$ 的位置, 再把 $h_{0 \dots n}$ 合并起来即可。

时间复杂度 $O(n^2 2^n)$ 。

占位多项式

考虑构造占位多项式 $w(x) = |x|$, 我们把两个条件拆成 $x \cup y = i$ 与 $|x| + |y| = |i|$ 。

把所有 $|x| = k$ 的 A_x 分成一组, 记为 f_k (这是一个长为 2^n 的序列, 其中满足 $|x| = k$ 的位置 f_x 的值为 $f_x = A_x$, 其余为 $f_x = 0$), 把 B 那边做相应的分组, 设为 g 。

那么只需要做卷积 $f_i \star g_j \rightarrow h_{i+j}$ (其中 \star 表示两个序列的或卷积), 最后只保留 h_k 中 $|x| = k$ 的位置, 再把 $h_{0 \dots n}$ 合并起来即可。

时间复杂度 $O(n^2 2^n)$ 。

可以把这两维理解成二元幂级数, 即 $F(x, y) = \sum f_i x^i y^{w(i)}$, 那么子集卷积相当于对第一维进行或卷积, 第二维进行和卷积, 即定义 $x^a y^b \times x^c y^d \rightarrow x^{a \cup b} y^{c+d}$ 。

当然第一维写成 xor 卷积也没问题, 但有一个缺点是需要 2 存在逆元。

Luogu3349. 小星星

给定一张 n 点 m 边的无向图 $G = (V_G, E_G)$ 与一棵 n 个节点的树 $T = (V_T, E_T)$ 。 $(V_G = V_T = \{1, 2, \dots, n\})$

问有多少个 $1, 2, \dots, n$ 的排列 p 满足：对每个 $(i, j) \in E_T$ ，均有 $(p_i, p_j) \in E_G$ 。

$$1 \leq n \leq 17, 1 \leq m \leq \frac{n(n-1)}{2}.$$

小星星

考虑 DP。

设 $f(u, j, S)$ 表示填完了 u 的子树，已经用了 S 以内的这些数，并且 $p_u = j$ 的方案数。

每次加进来一个子树的时候，只需要做一次 FMT/FWT，然后只保留 popcount 恰好为当前总 size 的位置即可。

时间复杂度 $O(n^3 2^n)$ 。

半半在线卷积

子集卷积可以用来优化一些形如 $\text{dp}(S) = \sum_{T \subset S} \text{dp}(T)g(S - T)$ 的 DP 转移方程。

类似普通的子集卷积，我们写出 $n + 1$ 个长为 2^n 的序列 $f_0 \dots f_n$ ，其中

$$f_{i,S} = \begin{cases} \text{dp}(S) & , |S| = i \\ 0 & , \text{otherwise.} \end{cases}$$

注意到 $|S| = k$ 的所有 $\text{dp}(S)$ 只会从 $|T| < k$ 的 $\text{dp}(T)$ 中转移过来，因此考虑按照 $|S|$ 从小到大转移，每次用已经算出的 $f_0 \dots f_k$ 去计算 f_{k+1} 。时间复杂度 $O(n^2 2^n)$ 。

如果我们试图按照集合的二进制数从小到大进行计算半在线卷积，则需要使用「全半在线卷积」；而这种算法称为「半半在线卷积」。

Luogu4221. 州区划分

给定一张 n 点 m 边的无向图，第 i 个点有点权 w_i 。

现需要将这 n 个点划分为若干个有序的组，设 V_i 为第 i 个组的点集，那么每个 V_i 不能为空且不能存在欧拉回路。

定义第 i 个组的满意度为第 i 个组的权值和在前 i 个组的权值和中所占比例的 p 次幂，即

$$\left(\frac{\sum_{x \in V_i} w_x}{\sum_{j=1}^i \sum_{x \in V_j} w_x} \right)^p$$

定义一个划分的满意度为所有组的满意度的乘积。求所有合法的划分方案的满意度之和对 998244353 取模的值。

$1 \leq n \leq 21, 0 \leq m \leq \frac{n(n-1)}{2}, 1 \leq w_i \leq 100, 0 \leq p \leq 2$, 时限 10s。

州区划分

用 $c(S) = 0/1$ 表示 S 是否合法，这个可以 $O((n+m)2^n)$ 算出。

$w(S)$ 表示 S 以内点权和， $\text{dp}(S)$ 表示给点集 S 分组的答案，那么有转移

$$\text{dp}(S) = \sum_{T \subset S} \text{dp}(T) \times c(S - T) \left(\frac{w(S - T)}{w(S)} \right)^p$$

使用半半在线卷积优化，复杂度 $O(n^2 2^n)$ 。

Luogu6570. 优秀子序列

给定一个长度为 n 的非负整数序列 $A = \{a_1, a_2, \dots, a_n\}$, 对于 A 的一个子序列

$B = \{a_{b_1}, a_{b_2}, \dots, a_{b_m}\}$ ($0 \leq m \leq n$, $1 \leq b_1 < b_2 < \dots < b_m \leq n$, 下同), 称 B 是 A 的优秀子序列当且仅当, 其任意两个不同元素的按位与结果均为 0, 即: $\forall 1 \leq i < j \leq m$, 满足:
 $a_{b_i} \text{ and } a_{b_j} = 0$, 其中 and 是按位与运算。

对于子序列 $B = \{a_{b_1}, a_{b_2}, \dots, a_{b_m}\}$, 我们定义其价值为 $\varphi(1 + \sum_{i=1}^m a_{b_i})$, 其中 $\varphi(x)$ 表示小等于 x 的正整数中与 x 互质的数的个数。

现在请你求出 A 的所有优秀子序列的价值之和, 答案对 $10^9 + 7$ 取模。

$1 \leq n \leq 10^6$, $0 \leq a_i \leq 2 \times 10^5$ 。

优秀子序列

考虑对每个 S 算出 $f(S)$ 表示选出若干个数，使得其不交并恰好为 S 的方案数。

优秀子序列

考虑对每个 S 算出 $f(S)$ 表示选出若干个数，使得其不交并恰好为 S 的方案数。

考虑 DP，设 $g(S) = \sum_{i=1}^n [a_i = S]$ ，由于这里是无序的，考虑枚举最小元所在集合，有转移

$$f(S) = \sum_{\min(S) \in T \subseteq S} f(S - T)g(T)$$

优秀子序列

考虑对每个 S 算出 $f(S)$ 表示选出若干个数，使得其不交并恰好为 S 的方案数。

考虑 DP，设 $g(S) = \sum_{i=1}^n [a_i = S]$ ，由于这里是无序的，考虑枚举最小元所在集合，有转移

$$f(S) = \sum_{\min(S) \in T \subseteq S} f(S - T)g(T)$$

考虑按照最小元从大到小转移，每次用最小元 $> i$ 的集合做子集卷积算出最小元 $= i$ 的集合的 DP 值，总复杂度是

$$\sum_{i=1}^n i^2 2^i = O(n^2 2^n)$$

Luogu U280360. 天空度假山庄

云浅与 Yoimiya 一起来到了「天空度假山庄」。「天空度假山庄」中有一个 n 点 m 边的无向图 G , 图中点的编号分别为 $1, 2, \dots, n$, 边的编号分别为 $1, 2, \dots, m$ 。

度假山庄已经很久没有打扫了, 因此图中的边已经模糊不清。具体来说:

- 第 i 条边连接的两个端点分别为 u_i 和 v_i 。
- 这条边上落下了许多灰尘, 有 p_i 的概率直接消失。这里保证 $0 < p_i < 1$ 。

对于图 G , Yoimiya 定义 $f(G)$ 为图中连通块个数的期望值; 同时, 她定义 $G[l : r]$ 表示 $G = (V, E)$ 的一个子图 $G[l : r] = (V', E')$, 其中

$$V' = \{l, l+1, \dots, r\}$$
$$E' = \{(u, v) | u \in V', v \in V', (u, v) \in E\}$$

对每个 $1 \leq l \leq r \leq n$, 你都需要帮助 Yoimiya 求出 $f(G[l : r])$ 对 998244353 取模的值。

$1 \leq n \leq 19, 1 \leq m \leq \frac{n(n-1)}{2}$, 时限 6s。

天空度假山庄

考慮 DP，設 $f(S)$ 表示只考慮 S 以內點和邊，且最終 S 內所有點聯通的概率。

天空度假山庄

考虑 DP，设 $f(S)$ 表示只考虑 S 以内点和边，且最终 S 内所有点联通的概率。

记 $x = \min(S)$ 为集合 S 中的最小值，考虑容斥：若 S 内所有点不连通，枚举 x 所在的连通块 $x \in T \subset S$ ，有转移

$$f(S) = 1 - \sum_{x \in T \subset S} f(T)p(T, S - T)$$

其中 $p(A, B)$ 表示一端点在 A 内，一端点在 B 内的所有边均消失的概率，即

$$p(A, B) = \sum_{u_i \in A, v_i \in B} \frac{x_i}{x_i + y_i} \quad (A \cap B = \emptyset)$$

天空度假山庄

接下来，再设 $g(S)$ 表示只考虑 S 内点和边，连通块个数的期望值。仍然只需要枚举 $\min(S)$ 所在连通块。具体来说，有

$$g(S) = 1 + \sum_{\min(S) \in T \subset S} f(T)p(T, S - T)g(S - T)$$

这样就有一个 $O(3^n)$ 的算法。

天空度假山庄

注意到如果设 $w(S)$ 表示两端点都在 S 内的边全部消失的概率，那么

$$p(A, B) = \frac{w(A \cup B)}{w(A) \times w(B)}$$

天空度假山庄

注意到如果设 $w(S)$ 表示两端点都在 S 内的边全部消失的概率，那么

$$p(A, B) = \frac{w(A \cup B)}{w(A) \times w(B)}$$

因此转移方程可以进一步写成

$$\frac{f(S)}{w(S)} = \frac{1}{w(S)} - \sum_{\min(S) \in T \subset S} \frac{f(T)}{w(T)} \times \frac{1}{w(S-T)}$$

$$\frac{g(S)}{w(S)} = \frac{1}{w(S)} + \sum_{\min(S) \in T \subset S} \frac{f(T)}{w(T)} \times \frac{g(S-T)}{w(S-T)}$$

天空度假山庄

注意到如果设 $w(S)$ 表示两端点都在 S 内的边全部消失的概率，那么

$$p(A, B) = \frac{w(A \cup B)}{w(A) \times w(B)}$$

因此转移方程可以进一步写成

$$\frac{f(S)}{w(S)} = \frac{1}{w(S)} - \sum_{\min(S) \in T \subset S} \frac{f(T)}{w(T)} \times \frac{1}{w(S-T)}$$

$$\frac{g(S)}{w(S)} = \frac{1}{w(S)} + \sum_{\min(S) \in T \subset S} \frac{f(T)}{w(T)} \times \frac{g(S-T)}{w(S-T)}$$

按照 $\min(S)$ 从大到小转移即可，复杂度 $O(n^2 2^n)$ 。

集合幂级数的多项式操作

由于子集卷积比 and/or/xor 卷积常用得多，因此我们这里只考虑子集卷积定义下的集合幂级数多项式操作。

注意到把子集卷积拆成二元形式幂级数之后，第一维与第二维是相当独立的。

集合幂级数的多项式操作

也就是说，如果我们先按照 `popcount` 分类成 $f_{0 \dots n}$ ，接下来把每个 f_i 都做 FMT，那么只需要对 f 做正常的多项式操作（此时 $f_i + f_j$ 定义为对应位置相加， $f_i \times f_j$ 定义为对应位置相乘，注意 f_i 是一个长为 2^n 的序列），最后再 IFMT 之后合并。

因此，如果我们对长为 n 的多项式做某个多项式操作的复杂度为 $M(n)$ ，那么对 $2^{[n]}$ 上的集合幂级数做多项式操作的复杂度就是 $O(n^2 2^n + M(n) 2^n)$ 。

求逆/ln/exp

由于

$$\frac{1}{1 - F(x)} = \sum_{i \geq 0} (F(x))^i$$

因此，求逆可以看作有序地进行集合组合。

求逆/ln/exp

由于

$$\frac{1}{1 - F(x)} = \sum_{i \geq 0} (F(x))^i$$

因此，求逆可以看作有序地进行集合组合。

$$e^{F(x)} = \sum_{i \geq 0} \frac{(F(x))^i}{i!}$$

那么 \exp 就是无序地进行集合组合， \ln 就是逆变换。

求逆/ln/exp

由于

$$\frac{1}{1 - F(x)} = \sum_{i \geq 0} (F(x))^i$$

因此，求逆可以看作有序地进行集合组合。

$$e^{F(x)} = \sum_{i \geq 0} \frac{(F(x))^i}{i!}$$

那么 \exp 就是无序地进行集合组合， \ln 就是逆变换。

上面的《优秀子序列》，《天空度假山庄》两道题就分别对应了集合幂级数的 \exp 与 \ln 。（其中《优秀子序列》中还需要特殊处理 $a_i = 0$ 的项）实际上你也可以认为《黎明前的巧克力》等题目可以对应异或卷积下的 \exp 。

$O(n^2)$ 求逆

$F(x)G(x) = \epsilon \Rightarrow g_0 = \frac{1}{f_0}$ (这里 ϵ 代表子集卷积单位元, 1 可以认为是一个长为 2^n 但只有第 0 项是 1 的序列), 以及

$$\sum_{j=0}^i f_{i-j}g_j = 0 \iff g_i = -\frac{1}{f_0} \sum_{j=0}^{i-1} f_{i-j}g_j \quad (i > 0)$$

由于在子集卷积中我们只关心 f_i 中 $\text{popcount } i = i$ 的项, 因此集合幂级数存在逆只需要 f_0 的第 0 项不为 0, 即原集合幂级数的 x^\varnothing 项不等于 0。

$O(n^2)$ exp

$G(x) = e^{F(x)}$, 两边求导得到 $G'(x) = F'(x)e^{F(x)} = F'(x)G(x)$, 比较
两边系数得到

$$g_0 = \exp f_0$$

$$ig_i = [x^{i-1}]G'(x) = [x^{i-1}]F'(x)G(x) = \sum_{j=0}^{i-1} f_{i-j}(i-j)g_j \quad (i > 0)$$

由此我们可以知道存在 exp 需要 x^\emptyset 项 = 0 (否则 g_0 无意义), 从组合意义上也可以理解, 若空集的方案不为 0, 我们可以往里放任意多个空集。

$O(n^2) \ln$

$F(x) = \ln G(x)$, 首先要有 $g_0 = 1$ 以及 $f_0 = 0$ 。可以理解为 \ln 是 \exp 的逆变换, 存在 \exp 需要 x^\varnothing 项为 0, 这意味着 \exp 完之后总有 x^\varnothing 项为 1, 也就意味着存在 \ln 需要 x^\varnothing 项为 1。

$O(n^2) \ln$

$F(x) = \ln G(x)$, 首先要有 $g_0 = 1$ 以及 $f_0 = 0$ 。可以理解为 \ln 是 \exp 的逆变换, 存在 \exp 需要 x^\varnothing 项为 0, 这意味着 \exp 完之后总有 x^\varnothing 项为 1, 也就意味着存在 \ln 需要 x^\varnothing 项为 1。

把 \exp 的式子反过来, 由

$$ig_i = \sum_{j=0}^{i-1} f_{i-j}(i-j)g_j \iff g_i = \frac{1}{i}f_i \times i + \frac{1}{i} \sum_{j=1}^{i-1} f_{i-j}(i-j)g_j$$

可得

$$f_i = g_i - \frac{1}{i} \sum_{j=1}^{i-1} f_{i-j}(i-j)g_j$$

尽管这样做 \exp/\ln 和上面那种根据最小元按顺序转移的复杂度相同, 但常数小得多, 写起来也更简单。

Luogu6846. Amusement Park

有一个含 n 个点， m 条边的有向图，图无重边，无自环，两点之间不成环。

现在我们想改变一些边的方向，使得该有向图无环。

您需要求出，每一种改变方向后使得该有向图无环的方案的需改变边的数量之和 $\text{mod } 998244353$ 之后的答案。

$$1 \leq n \leq 18, 0 \leq m \leq \frac{n(n-1)}{2}.$$

Amusement Park

首先发现，对于一种合法方案，把边反向后仍然合法，且两种方案的反转边数之和为 m 。因此只需要算方案数乘上 $\frac{m}{2}$ 即可。

Amusement Park

首先发现，对于一种合法方案，把边反向后仍然合法，且两种方案的反转边数之和为 m 。因此只需要算方案数乘上 $\frac{m}{2}$ 即可。

考虑设 $F(S)$ 表示把点集 S 以及点集内边定向的方案，转移时钦定集合 T 内的点入度为 0，不难得到容斥系数为

$$\sum_{\emptyset \subset C \subseteq T} (-1)^{|T|-|C|} = \sum_{i=0}^{|T|-1} \binom{|T|}{i} (-1)^i = (-1)^{|T|-1}$$

于是有转移方程

$$F(S) = \sum_{\emptyset \subset T \subseteq S} (-1)^{|T|-1} c(T) F(S - T)$$

其中 $c(S) \in \{0, 1\}$ 表示点集 S 中是否存在边。

Amusement Park

设 $G(S) = (-1)^{|S|-1} c(S)$, 规定 $G(\emptyset) = 0, F(\emptyset) = 1$ 。

此时你可以使用半半在线卷积优化, 不过进一步我们发现这个式子就是 $F = F \cdot G + 1 \iff F = \frac{1}{1-G}$ (+1 是因为 $F(\emptyset) = 1$), 因此直接求逆即可。

二者的常数貌似差别不大, 没有 \exp/\ln 与半半在线卷积的差距那么大。

胡策的统计

对于一个无向图定义它的连通值为该图连通块数的阶乘。

给定一个 n 个结点的简单无向图 G , 求 G 的所有 2^m 个生成子图的连通值之和。

生成子图指的是保留所有点, 但只保留边集的一个子集得到的新图。

$$1 \leq n \leq 20, 0 \leq m \leq \frac{n(n-1)}{2}.$$

胡策的统计

首先做 \ln 求出 $F(x)$ 表示一个集合以内点联通的方案数。

接下来把阶乘看作有序地进行集合组合，发现就是求逆。

答案就是 $[x^U] \frac{1}{1-F(x)}$ 。复杂度 $O(n^2 2^n)$ 。

QOJ5019. 整数

计算机内有一个长度为 n 的非负整数数组 a ，下标为 $i(0 \leq i < n)$ 的元素取值范围为 $[0, R_i]$ 。

由于某些特殊原因，若这 n 个数的同一个二进制位上的取值出现某些组合则计算机会炸掉。

具体地，设 $b_x = \sum_{i=0}^{n-1} (\lfloor \frac{a_i}{2^x} \rfloor \bmod 2)$ ，那么只有当 $\forall x \geq 0, b_x \in S$ 时计算机不会炸。其中 S 是某个给定的集合。

求能保证机子安全的数组取值的方案数，答案对 998244353 取模。

$1 \leq n \leq 18, 0 \leq R_i < 2^{60}, 0 \in S$ 。

整数

$f(i, S)$ 表示从高到低填到第 i 位，目前已经不顶上界的位置集合为 S 的方案数。

整数

$f(i, S)$ 表示从高到低填到第 i 位，目前已经不顶上界的位置集合为 S 的方案数。

设 $g(S) \in \{0, 1\}$ 表示 S 这个数是否被禁止，那么转移就相当于给 $f(i, \cdot)$ 和 g 做卷积。

对每一位讨论一下 R_i 是 0 还是 1，即可确定这一位的转移矩阵，做 FWT 即可。

整数

$f(i, S)$ 表示从高到低填到第 i 位，目前已经不顶上界的位置集合为 S 的方案数。

设 $g(S) \in \{0, 1\}$ 表示 S 这个数是否被禁止，那么转移就相当于给 $f(i, \cdot)$ 和 g 做卷积。

对每一位讨论一下 R_i 是 0 还是 1，即可确定这一位的转移矩阵，做 FWT 即可。

可以提前 FWT 一手，最后再还原。复杂度 $O(n2^n \log R)$ 。

Luogu9131. Problem Setting P

Farmer John 出了 n 道题，聘了 m 个验题人来品鉴难度。

难度只有简单和困难两种。

Farmer John 将从中选出若干道（至少一道），并以一定顺序排列，使得前一道题目中任意一个觉得此题困难的验题人也觉得后面一道题目困难。

回答有多少种选出来并排列的方案，对 $10^9 + 7$ 取模。

$1 \leq n \leq 10^5$, $1 \leq m \leq 20$ 。要求时间复杂度 $O(m2^m)$ 。

Problem Setting P

考虑 DP, $f(S)$ 表示以 S 结尾的方案数。首先把相同的压到一起, 那么转移就是

$$f(S) = \left(\sum_{i=1}^{c(S)} \binom{c(S)}{i} i! \right) \times \sum_{T \subset S} f(T)$$

Problem Setting P

考虑 DP, $f(S)$ 表示以 S 结尾的方案数。首先把相同的压到一起, 那么转移就是

$$f(S) = \left(\sum_{i=1}^{c(S)} \binom{c(S)}{i} i! \right) \times \sum_{T \subset S} f(T)$$

首先把前面一项预处理。此时可以直接半半在线卷积, 但复杂度 $O(m^2 2^m)$ 。

Problem Setting P

考虑 DP, $f(S)$ 表示以 S 结尾的方案数。首先把相同的压到一起, 那么转移就是

$$f(S) = \left(\sum_{i=1}^{c(S)} \binom{c(S)}{i} i! \right) \times \sum_{T \subset S} f(T)$$

首先把前面一项预处理。此时可以直接半半在线卷积, 但复杂度 $O(m^2 2^m)$ 。

考虑 FMT 的同时转移: 设 $g(S, k)$ 表示所有满足 $T \subseteq S$ 且 S, T 只有前 k 位可能不同的所有 $f(T)$ 之和。那么已知所有 $T \subseteq S$ 的 $f(T)$ 和 $g(T, \cdot)$, 可以 $O(M)$ 递推出所有 $g(S, 0 \cdots M - 1)$ 的值; 已知所有 $S \subset T$ 的 $g(S, \cdot)$ 之后, 枚举 T 与 S 的第一个不同的位转移即可。

CF1773G. Game of Questions

Genie 正在参加一个问答比赛。比赛共 n 题，有 m 个参赛者（Genie 为 1 号参赛者）。

比赛的形式如下：先将 n 道题随机排序（即每个排列出现的概率都是 $\frac{1}{n!}$ ），然后按排列的顺序会依次问出这 n 个问题。问一个问题时，若所有人都会或所有人都不会则无事发生，否则不会的人会被淘汰。在 n 个问题都被问完之后，未被淘汰的人就都赢得胜利。

现在给出每个人是否会每道题（即一个 $n \times m$ 的 01 矩阵），请求出 Genie 获胜的概率。

原题： $n \leq 2 \times 10^5, m \leq 17$ 。加强版： $m \leq 22$ 。

Sol

注意到一个题目被问两次是没有意义的，因此我们可以把这个过程看作每次随机取一个问题（可能已经被问过了），问最后每个选手获胜的概率。

那么就可以以当前未被淘汰的选手作为状态，得到一张 2^m 个点的状态图，每个点有 n 条出边，每个点处可能有自环，除此之外是有向无环的。考虑在上面直接 DP，可以只处理有效的转移边，做到 $O(3^m)$ 。

Sol

考虑所有有效的转移，发现一定是由 popcount 大的转移到 popcount 严格小的。

设 A_i 为会做第 i 道题的人的集合。

于是我们可以枚举 $i = m, m - 1, \dots, 0$ ，计算 $\text{popcount} > i$ 状态的对 $\text{popcount} = i$ 的状态的贡献。这个贡献形如 $\frac{dp_S}{c_S} \times g_T \rightarrow dp_{S \cap T}$ ，其中 c_S 是 A_i 和 S 有交但不包含 S 的题目 i 的数量， g_T 是 $A_i = T$ 的 i 的数量，这里要求 $0 < |S \cap T| < |S|$ 。

这可以通过一遍高维前缀差分与高维前缀和来算出，于是我们就做到了 $O(m^2 2^m)$ 。

CF1034E. Little C Loves 3 III

给两个长为 2^n 的序列 a, b , 要求做 mod4 意义下的子集卷积。

$$n \leq 21$$

$n \leq 21$ 不允许我们多加一个占位符，必须要做到 $O(n2^n)$ 。

那怎么办呢，发现我们只需要重新令 $A_i = 4^{|i|} \times a_i$, B_i 类似，然后对 A, B 做 OR 卷积（必须是 OR），最后把结果的第 i 项除掉 $4^{|i|}$ 后，再对 4 取模即可。

原理：考虑所有贡献到 k 的 i, j ，必有 $|i| + |j| \geq |k|$ ，那么严格大于的部分除掉 $4^{|k|}$ 再对 4 取模后就没了。

时间复杂度 $O(n2^n)$ 。

UOJ328. 量子破碎

题面比较复杂，建议去看原题。

可以用 n 次操作做一遍 FWT，假设得到 b ，那么

$$b_k = \frac{(-1)^{|x \& k|} + (-1)^{|y \& k|}}{2^{(n+1)/2}}.$$

也就是说，返回的 k 一定满足 $|x \& k|, |y \& k|$ 奇偶性相同，即 $|(x \oplus y) \& k| = 0$ 。

那么相当于每次可以返回一个关于 $x \oplus y$ 的线性方程组，期望操作轮数（注意 $x \oplus y \neq 0$ ）可以计算出来，就是 $1 + \sum_{i=0}^{n-2} \frac{2^{n-1}}{2^{n-1}-2^i} \leq n+2$ 于是总操作次数期望意义下不超过 $n^2 + 2n$ 。

CF1336E. Chiori and Doll Picking

给定一个长为 n 的序列 a , 其中 $0 \leq a_i < 2^m$ 。

对 $i = 0, 1, \dots, m$ 求出: 选一个子序列, 使得其异或和的 `popcount` 恰好为 i 的方案数。

$1 \leq n \leq 2 \times 10^5, 1 \leq m \leq 35$ 。

Hard version: $m \leq 53$ 。

首先求一个线性基 B , 里面至多有 m 个数。对这 $|B|$ 个数求答案再把答案乘上 $2^{n-|B|}$ 即可。

Easy version: 先把线性基消元使得所有数最高位不同, 把数按照最高位 $\geq m/2$ 和 $< m/2$ 分类, 对于最高位低于 $m/2$ 的直接 $2^{m/2}$ 算出 $f_S = 0/1$ 表示异或能否 S , 对于最高位 $> m/2$ 的 $O(2^{m/2})$ 算出 $g_{i,S}$ 表示选一个子集求异或和, 其 $> m/2$ 的位上共有 i 个 1, $\leq m/2$ 的位上为 S 的方案数。

最后枚举 i , 把 $g_{i,\dots}$ 和 f_{\dots} 做 xor 卷积即可。复杂度 $O(m^2 2^{m/2})$ 。
当然做法比较多, 也可以不用 FWT 得到一个 $O(2^{m/2} m^2)$ 做法。

Hard version：首先 $|B| \leq m/2$ 时可以暴力枚举。考虑 $|B| > m/2$ 怎么做。

设 $B = \{x_1, \dots, x_k\}$, 设 $f_i(t) = 1 + t^{x_i}$, 我们希望统计 $\prod f_i(t)$ 在每个 `popcount` 上的元素和。

做 FWT, 有 $\text{FWT}(f_i(t))_j = 1 + (-1)^{|j \& x_i|}$, 最后 IFWT 回来, 得到

$$[t^u] \prod f_i(t) = \frac{1}{2^m} \sum_{j=0}^{2^m-1} (-1)^{|u \& j|} \prod_{i=1}^n (1 + (-1)^{|j \& x_i|})$$

我们发现乘积项中只要出现 0 那就没有贡献了，而想要乘积项中没有 0，必须要所有 x_i 都满足 $|j \& x_i|$ 为偶数。这样的 j 的数量是多少？看作 \mathbb{F}_2 上的一个线性方程组，且方程之间是线性无关的，那么个数恰好为 $2^{n-|B|}$ 。

于是，对每个 u ，只有 $2^{n-|B|}$ 个 j 会造成贡献。枚举 j ，计算它对所有 u 的贡献即可。（注意这只和 j 的 `popcount` 相关）

结合两部分，我们得到一个 $O(2^{m/2})$ 的做法。

ABC288Ex. A Nameless Counting Problem

给定 N, M, X , 问有多少长为 N 的非负整数序列 A 满足:

- $0 \leq A_1 \leq A_2 \leq \cdots \leq A_N \leq M$ 。
- $A_1 \oplus A_2 \oplus \cdots \oplus A_N = X$ 。

原题范围: $1 \leq N \leq 200, 0 \leq M, X < 2^{30}$; 加强版: $1 \leq N \leq 5 \times 10^5$ 。

考虑列生成函数： t 表示 xor 卷积的元， x 来刻画选的个数为，即定义 $t^S x^i \times t^T x^j = t^{S \text{ xor } T} x^{i+j}$ ，那么答案就是

$$[t^X x^N] \prod_{i=0}^M (1 + t^i x + t^2 + t^3 x + \dots) = [t^X x^N] \prod_{i=0}^N \left(\frac{1}{1-x^2} + \frac{x}{1-x^2} t^i \right)$$

设 $f_i(t) = 1 + t^i x + t^2 + \dots$ ，我们考虑做 FWT，有

$$\text{FWT} \left(\frac{1}{1-x^2} + \frac{x}{1-x^2} t^i \right)_j = \frac{1}{1-x^2} + (-1)^{|i| \text{ and } j} \frac{x}{1-x^2}$$

设 $2^{D-1} \leq M < 2^D$ 。我们再把 FWT 全部点乘，做 IWT：

$$[t^X] \prod_{i=0}^M (1 + t^i x + t^{2i} + \dots) = \frac{1}{2^D} \sum_{j=0}^{2^D-1} (-1)^{|X \text{ and } j|} \prod_{i=0}^M \text{FWT}(f_i)_j$$

那么关键在于计算 $\prod_{i=0}^M \text{FWT}(f_i)_j$ 。

注意到这只会是 $\frac{1}{1-x^2} + \frac{x}{1-x^2} = \frac{1}{1-x}$ ，或者 $\frac{1}{1-x^2} - \frac{x}{1-x^2} = \frac{1}{1+x}$ ，那么只要我们能对 j 求出 $[0, M]$ 中有多少个 i 使得 $|i \text{ and } j|$ 是奇数 or 偶数，就能快速知道 c_j ，意思是 $\prod_{i=0}^M \text{FWT}(f_i)_j = \frac{1}{(1+x)^{c_j} (1-x)^{M-c_j}}$ 。

不妨先来考虑 $M = 2^D - 1$ 的情形，我们发现此时只要 j 的二进制位有任何一个 1，改变 i 在这一位上的值总可以改变 $|i \text{ and } j|$ 的奇偶性，即 $j \neq 0$ 时恰好有 2^{D-1} 个 i 使得 $|i \text{ and } j|$ 为偶数，剩下的是奇数；而 $j = 0$ 时自然是偶数。

那么进一步，对确定的 M ，考虑枚举 i 和 M 的第一个不同的位，假设是第 p 位 (M 在第 p 位上是 1， i 和 M 在更高位上相同，但 i 这一位是 0)，那么 i 后面是任取的，类似地，只要 j 在这后面的位有至少一个 1，那么就是对半分；否则是全都为偶数。

综上所述，我们得到结论：所有 c_j 只有至多 $2 \log M$ 种，且我们可以
通过枚举 j 的 lowbit 快速得到这些 c_j 和对应的 j 的个数。那么现在只
剩下给定 c ，计算 $[x^N] \frac{1}{(1+x)^c (1-x)^{M-c}}$ ，这个直接 $O(N)$ 计算就好了。
总复杂度 $O(N \log M)$ 。

练习：CF2096H

AGC034F. RNG and XOR

给定 n 和一个长度为 2^n 的数组 A (从 0 标号).

有一个初始为 0 的变量 x . 不断操作, 每次操作以 $\frac{A_i}{\sum_{j=0}^{2^n-1} A_j}$ 的概率

将 x 变成 $x \oplus i$.

对于所有 $i \in [0, 2^n)$, 求出 x 第一次变成 i 的期望操作次数.

$n \leq 18, 1 \leq A \leq 1000$.

Solution

<https://www.luogu.com.cn/article/ug8qq7>

Thanks for listening

