Fast Walsh-Hadamard Transform

SGColin

目录

1 Fast Walsh-Hadamard Transform

用于解决二元逻辑运算卷积的方法。

$$C = A \oplus B \ : \ C_i = \sum_{j \oplus k = i} A_j \times B_k$$

其中 $\oplus$ 指任意二元逻辑运算，常见的 OR , AND , XOR ，以及三种运算的非形式。

因为运算的需要，我们默认 A, B 都是长度 n 为 2^K 的向量。

先定义一些东西。

$$A \pm B = (\ A_0 \pm B_0\ ,\ A_1 \pm B_1\ ,\ \dots\ ,\ A_{n-1} \pm B_{n-1}\ )$$

$$A \oplus B = (\ \sum_{j \oplus k = 0} A_j \times B_k\ ,\ \sum_{j \oplus k = 1} A_j \times B_k\ ,\ \dots\ ,\ \sum_{j \oplus k = n-1} A_j \times B_k\ )$$

1.1 基础的想法

首先暴力是 $\mathrm{O}(n^2)$ 的，所以需要进行优化。

思路来自位运算各个二进制位是独立的。我们先**只考虑最高的二进制位**。

$$A = (A_0, A_1)\ , B = (B_0, B_1)$$

其中 A_0 表示下标最高位为 0 的项对应的向量，也就是将向量 A 前一半记作 A_0，类比的定义 A_1 ，将向量 A 后一半记作 A_1。

因此我们可以得到该表示下的对应关系：

$$C = A \oplus B = (\ \sum_{j \oplus k = 0} A_j \times B_k\ ,\ \sum_{j \oplus k = 1} A_j \times B_k\ )$$

OR 运算下，卷积满足 $C = A \oplus B = (A_0 \oplus B_0, A_0 \oplus B_1 + A_1 \oplus B_1)$

AND 运算下，卷积满足 $C = A \oplus B = (A_1 \oplus B_0 + A_0 \oplus B_1, A_1 \oplus B_1)$

XOR 运算下，卷积满足 $C = A \oplus B = (A_0 \oplus B_0 + A_1 \oplus B_1, A_0 \oplus B_1 + A_1 \oplus B_0)$

可以通过展开证明定义在向量上的 $\oplus$ 运算满足交换律、结合律和**加法连接分配律**。

类似 FFT 的思想，如果我们构造变换 tf 满足

$$tf(A) \times tf(B) = tf(C)\ \ (\times \text{ 的含义就是对应项相乘})$$

那么如果改变换 tf 和其逆变换 utf 都能快速实现，复杂度就可以优化了。

下面我们根据不同的运算给出 tf 与 utf 的具体情况。

关于验证 $utf(A)$ 很方便，直接代入得 $utf(tf(A)) = A$ 即可证明。

1.2 具体的变换

1.2.1 OR

$$tf(A) = (tf(A_0), tf(A_0) + tf(A_1))$$

$$utf(A) = (utf(A_0), utf(A_1) - utf(A_0))$$

```
for (rg int i = 2; i <= n; i <<= 1)
  for (rg int p = i >> 1, j = 0; j < n; j += i)
    for (rg int k = j; k < j + p; ++k) s[k + p] += s[k]*op;
```

1.2.2 AND

$$tf(A) = (tf(A_0) + tf(A_1), tf(A_1))$$

$$utf(A) = (utf(A_0) - utf(A_1), utf(A_1))$$

```
for (rg int i = 2; i <= n; i <<= 1)
  for (rg int p = i >> 1, j = 0; j < n; j += i)
    for (rg int k = j; k < j + p; ++k) s[k] += s[k + p]*op;
```

1.2.3 XOR

$$tf(A) = (tf(A_0) + tf(A_1), tf(A_0) - tf(A_1))$$

$$utf(A) = (\frac{utf(A_0) + utf(A_1)}{2}, \frac{utf(A_0) - utf(A_1)}{2})$$

```
for (rg int i = 2; i <= n; i <<= 1)
  for (rg int p = i >> 1, j = 0; j < n; j += i)
    for (rg int k = j, x, y; k < j + p; ++k) {
      x = s[k]; y = s[k + p];
      s[k] = x + y; s[k + p] = x - y;
      if (op == -1) s[k] /= 2 , s[k + p] /= 2;
    }
```

1.2.4 NXOR , NAND , NOR

当运算为以上三种情况的非情况时，直接求原运算的结果，然后交换互反的两位上的值即可。

2 一些题目

2.1 [CSU 1911] Card Game

2.1.1 Description

有两个大小为 n 的数组，多次询问，每次给出一个 x ，询问从两个数组中各选一个数或起来答案为 x 的方案数。

2.1.2 Solution

开个两个计数器数组 a, b ，每次读入集合里的数就把对应的计数器累加一下。答案数组就是两个计数器数组的或卷积，直接 FWT 即可，询问是 $\mathrm{O}(1)$ 的。

2.2 [CodeForces 662C] Binary Table

2.2.1 Description

有一个 $n \times m$ ($n \leq 20, m \leq 10^5$) 的表格，每个格里面有一个 0/1，可以任意次将一行或者一列的 01 全部反转，问表格中最少有多少个 1 。

2.2.2 Solution

暴力的做法是枚举行操作的二进制状态，可以发现每一列在一个固定的行操作下，列是否操作的策略是固定的，预处理每一个异或后二进制的答案，复杂度为 $\mathrm{O}(m \times 2^n)$ 。

注意到相同的列状态对于同一个行操作集合的策略是相同的，所以我们记录 cnt_s 表示初始行状态为 s 的个数，记录 res_s 表示 s 状态该列操作自由时最小的 1 个数，显然有

$$res_s = \min(bitcount(s), n - bitcount(s))$$

其中 $bitcount(s)$ 表示 s 二进制下 1 的个数。

那么操作集合为 s 对应的答案为

$$ans_s = \sum cnt_{s_1} \times res_{s \oplus s_1} = \sum_{s_1 \oplus s_2 = s} cnt_{s_1} \times res_{s_2}$$

其中 $\oplus$ 表示 XOR 运算。

然后求 ans 数组的过程就可以用 FWT 优化了，复杂度 $\mathrm{O}(\log_2(2^n) \times 2^n) = \mathrm{O}(n \times 2^n)$。

2.3 [HDU 5909] Tree Cutting

2.3.1 Description

定义一个联通诱导子图的权，为其包含的所有点的权的异或和。

给出一棵 n $(n \le 10^3)$ 个节点的树，求权为 $0...m$ $(m \le 2^10)$ 的联通诱导子图分别有多少个。

2.3.2 Solution

点分的做法见另一个 pdf 。

先考虑暴力。设 $f[u][i]$ 表示，在 u 的子树中，包含 u 的连通块里，权为 i 的个数。

那么转移就是一个树形背包了，组合方式是异或，所以复杂度为 $O(nm^2)$。

$$f[u][i] = f[u][j] + \sum_{j \oplus k = i} f[u][j] * f[v][k]$$

前半部分是不取这个子树的方案，后半部分是取子树的方案。

实际上后半部分这个转移是异或卷积，所以直接上 FWT 优化即可。

2.4 [Topcoder SRM 518 Div1] Hard Nim

2.4.1 Description

两个人进行 Nim 游戏，现在想知道，如果这 n $(n \le 10^9)$ 堆石子满足每堆石子的初始数量是不超过 m $(m \le 5 \times 10^4)$ 的质数，后手能获胜的局面有多少种，答案对 $10^9 + 7$ 取模。

2.4.2 Solution

首先预处理出来 m 范围内的质数。

考虑进行一个简单的 DP ，$f[i][s]$ 表示有 i 堆石子，异或和为 s 的方案数，转移显然是按 i 分层的，可以发现这是一个异或卷积的形式。

$$f[i][s] = \sum_{j \oplus k = s} \mathrm{e}[j\ is\ prime] \times f[i-1][k]$$

我们设这个 0/1 数组为 a ，可以发现由 $f[0]$ 转移到 $f[n]$ 的过程中实际上是与 a 进行了 n 次异或卷积。由于这个运算是满足交换律的，所以我们可以先计算 a 自己的 $n-1$ 次异或卷积，并使用快速幂的思路加速计算。

由于 FWT 满足 $tf(a) \times tf(b) = tf(c)$ ，所以在计算 n 次 a 子集的卷积时，我们只需要 FWT 一次，然后将 $tf(a)$ 进行快速幂，最后在还原回去。

可以注意到由于 $f[0]$ 数组只有 $f[0][0] = 1$ ，所以与幂次卷积后也只有 $f[0][0] \times res[0]$ 会产生贡献，因此答案直接取 $res[0]$ 即可。