

FWT快速沃尔什变换

算法思想

或卷积

大概长这个样子 $C_k = \sum_{j=k} A_i \times B_j$

满足交换律、结合律 $(A+B)|C=A|C+B|C$ 均为显然

几个重要的结论：

$$\text{FWT}(A+B) = \text{FWT}(A) + \text{FWT}(B)$$

$\text{FWT}(A) = (\text{FWT}(A_0), \text{FWT}(A_0) + \text{FWT}(A_1))$ A_0 表示最高位为 \$0\$ 的部分，也就是前 2^{n-1} 项； A_1 表示最高位为 \$1\$ 的部分，也就是后 2^{n-1} 项。

对于 or 卷积来说 $\text{FWT}(A)[i] = \sum_{j|i=i} A[j]$

此时用数学归纳法有：

$$\text{FWT}(A|B)$$

$$= \text{FWT}((A|B)_0, (A|B)_1)$$

$$= \text{FWT}(A_0|B_0, A_0|B_1 + A_1|B_0 + A_1|B_1)$$

$$= (\text{FWT}(A_0|B_0), \text{FWT}(A_0|B_0 + A_0|B_1 + A_1|B_0 + A_1|B_1))$$

$$= (\text{FWT}(A_0) \times \text{FWT}(B_0), \text{FWT}(A_0) \times \text{FWT}(B_0) + \text{FWT}(A_0) \times \text{FWT}(B_1) + \text{FWT}(A_1) \times \text{FWT}(B_0) + \text{FWT}(A_1) \times \text{FWT}(B_1))$$

$$= (\text{FWT}(A_0) \times \text{FWT}(B_0), (\text{FWT}(A_0) + \text{FWT}(A_1)) \times (\text{FWT}(B_0) + \text{FWT}(B_1)))$$

$$= (\text{FWT}(A_0, \text{FWT}(A_0 + A_1)) \times (\text{FWT}(B_0, \text{FWT}(B_0 + B_1))))$$

$$= \text{FWT}(A) \times \text{FWT}(B)$$

得证

所以类似于 FFT 也有一个叫做 IFWT 的东西。

$$\text{IFWT}(A) = (\text{IFWT}(A_0), \text{IFWT}(A_1) - \text{IFWT}(A_0))$$

和卷积

和或卷积类似

结论：

\$FWT(A)=(FWT(A_{0})+A_{1}),FWT(A_{1}))\$

\$FWT(A+B)=FWT(A)+FWT(B)\$

同样的数学归纳法可以证明和卷积成立

\$IFWT(A)=(IFWT(A_{0})-IFWT(A_{1}),IFWT(A_{1}))\$

异或卷积

结论：

\$FWT(A)=(FWT(A_{0})+FWT(A_{1}),FWT(A_{0})-FWT(A_{1}))\$

\$FWT(A+B)=FWT(A)+FWT(B)\$

\$IFWT(A)=(\frac{IFWT(A_{0})-IFWT(A_{1})}{2},\frac{IFWT(A_{0})-IFWT(A_{1})}{2})\$

算法实现

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int MAXN=1500000;

int MOD=998244353;
int inv2=(MOD+1)>>1,N;
ll a1[MAXN],b1[MAXN],a2[MAXN],b2[MAXN],a3[MAXN],b3[MAXN];

void or_FWT(ll *P,int opt) {
    for(int i=2; i<=N; i<<=1)
        for(int p=i>>1,j=0; j<N; j+=i)
            for(int k=j; k<j+p; ++k)
                P[k+p]+=P[k]*opt,P[k+p]=(P[k+p]+MOD)%MOD;
}

void and_FWT(ll *P,int opt) {
    for(int i=2; i<=N; i<<=1)
        for(int p=i>>1,j=0; j<N; j+=i)
            for(int k=j; k<j+p; ++k)
                P[k]+=P[k+p]*opt,P[k]=(P[k]+MOD)%MOD;
}

void xor_FWT(ll *P,int opt) { //如果不是在模意义下的话，把逆元变成直接除二就好了。
    for(int i=2; i<=N; i<<=1)
        for(int p=i>>1,j=0; j<N; j+=i)
            for(int k=j; k<j+p; ++k) {
                ll x=P[k],y=P[k+p];
```

```

        P[k]=(x+y)%MOD;
        P[k+p]=(x-y+MOD)%MOD;
if(opt== -1)P[k]=1ll*P[k]*inv2%MOD,P[k+p]=1ll*P[k+p]*inv2%MOD;
    }
}

int main() {
    scanf("%d",&N);
    N=1<<N;
    for(int i=0;i<N;i++) {
        scanf("%lld",&a1[i]);
        a2[i]=a3[i]=a1[i];
    }
    for(int i=0;i<N;i++) {
        scanf("%lld",&b1[i]);
        b2[i]=b3[i]=b1[i];
    }
    or_FWT(a1,1);or_FWT(b1,1);
    for(int i=0;i<N;i++) a1[i]=a1[i]*b1[i]%MOD;
    or_FWT(a1,-1);
    and_FWT(a2,1);and_FWT(b2,1);
    for(int i=0;i<N;i++) a2[i]=a2[i]*b2[i]%MOD;
    and_FWT(a2,-1);
    xor_FWT(a3,1);xor_FWT(b3,1);
    for(int i=0;i<N;i++) a3[i]=a3[i]*b3[i]%MOD;
    xor_FWT(a3,-1);
    for(int i=0;i<N;i++) {
        printf("%lld ",a1[i]);
    }
    putchar(10);
    for(int i=0;i<N;i++) {
        printf("%lld ",a2[i]);
    }
    putchar(10);
    for(int i=0;i<N;i++) {
        printf("%lld ",a3[i]);
    }
    putchar(10);
    return 0;
}

```

代码练习

1.<https://www.luogu.com.cn/problem/CF1119H>

题目大意

给定 \$n\$ 个三元组，以及三个数 \$x,y,z\$ [每个三元组的内容为 \${a_i, b_i, c_i}\$] 表示这个组里面

有 x 个 a_i , y 个 b_i , z 个 c_i , 再给一个数 N , 让求对于 0 到 2^{N-1} 分别有多少种方案 , 使得在每个三元组中各选出一个数字 , 异或的结果等于这个数字 , 方案数对 998244353 取模。其中 $n \leq 10^5$, $N \leq 17$

题目解析

貌似可以把元组中的数字理解为某个多项式中 a_i 次幂的系数是 x , b_i 次幂的系数是 y , c_i 次幂的系数是 z , 之后对于 n 个多项式 , 顺次进行 FWT , 最后再 $IFWT$, 但是想法很美好 , 这样的复杂度是 $O(nk2^k)$ 显然是爆炸了。

我们再回去观察题目 , 为什么是三元组 , 也就是说一个多项式中只有三项的系数不为零 , 且都固定为 x,y,z 则 $FWT(F_k)[i] = c(i,a_k)x + c(i,b_k)y + c(i,c_k)z$ 其中 $c(i,j)$ 表示异或卷积从 j 到 i 的变换系数。而 $c(i,j) = (-1)^{cnt(i \& j)}$ 所以这个式子就是 $FWT(F_k)[i] = (-1)^{cnt(i \& a_k)}x + (-1)^{cnt(i \& b_k)}y + (-1)^{cnt(i \& c_k)}z$ 于是现在我们可以如此算出每一个式子的 FWT , 再求 $IFWT$ 但是这样的复杂度仍然是 $O(n+k)2^k$ 仍然是爆炸的。所以到这里还远没有结束。

我们现在要求的是 $S[i] = \prod_{k=1}^n FWT(F_k)[i]$ 将每一项的系数都求出来 , 最后再 $IFWT$ 就可以了。又因为对于等号右侧的每一项 , 一共只有八种可能 , 即 x,y,z 可正可负。当我们把每一项出现的次数求出 , 就可以利用快速幂求得此项答案 , 这样单独一项系数的复杂度就由 $O(n)$ 降为 $O(\log n)$ 就可以通过了。这里有一个小技巧 , 我们可以控制其中一项的系数为正 , 比如我们将每个元组的三个数都对 a 异或 , 这样元组变为 $(0, b_k \oplus a_k, c_k \oplus a_k)$ 我们最后再把结果的次数异或回去所有的 a_k 就是真正的次数了。这样所有的 a 被我们控制为 0 , 和任何的 i 取与都是 0 。于是八种可能变成了四种可能 , 即 $x+y+z, x+y-z, x-y+z, x-y-z$

我们分别设这四种出现的次数为 c_1, c_2, c_3, c_4 显然四者之和为 n 我们接下来用待定系数法的思想 , 比如我们把 y 重新取为 1 , 其余设为 0 。这样求一个 FWT 出来 , 对应 i 次方的系数是所有 $(-1)^{cnt(i \& b_k)}$ 的和。这个值是 $c_1 + c_2 - c_3 - c_4$ 同理可以对 z 操作一下 , 这样我们一共有三个式子。最后一个式子是令 $b_k \oplus c_k$ 的系数为 1 。于是这个就是前两种情况的卷积 , 又因为 FWT 是可以乘起来的 , 所以它就是 $(-1)^{cnt(i \& b_k)}(-1)^{cnt(i \& c_k)}$ 这个和对应的是 $c_1 - c_2 - c_3 + c_4$ 所以四个方程都有了 , 解个方程 , 最后快速幂一搞 , 最终复杂度是 $O((k+\log n)2^k)$ 可以通过。

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int MAXN=2000010;

int MOD=998244353;
int inv2=(MOD+1)>>1,N,n,x,y,z;
ll a1[MAXN],b1[MAXN],a2[MAXN],b2[MAXN],a3[MAXN],b3[MAXN];

ll quick_power(ll a,ll t,ll mod) {
    ll ans=1;
    while(t) {
        if(t&1)ans=ans*a%mod;
        a=a*a%mod;
        t>>=1;
    }
}
```

```

    return ans;
}

void or_FWT(ll *P,int opt) {
    for(int i=2; i<=N; i<<=1)
        for(int p=i>>1,j=0; j<N; j+=i)
            for(int k=j; k<j+p; ++k)
                P[k+p]+=P[k]*opt,P[k+p]=(P[k+p]+MOD)%MOD;
}

void and_FWT(ll *P,int opt) {
    for(int i=2; i<=N; i<<=1)
        for(int p=i>>1,j=0; j<N; j+=i)
            for(int k=j; k<j+p; ++k)
                P[k]+=P[k+p]*opt,P[k]=(P[k]+MOD)%MOD;
}

void xor_FWT(ll *P,int opt) {
    for(int i=2; i<=N; i<<=1)
        for(int p=i>>1,j=0; j<N; j+=i)
            for(int k=j; k<j+p; ++k) {
                ll x=P[k],y=P[k+p];
                P[k]=(x+y)%MOD;
                P[k+p]=(x-y+MOD)%MOD;
    if(opt== -1) P[k]=1ll*P[k]*inv2%MOD,P[k+p]=1ll*P[k+p]*inv2%MOD;
            }
}

int main() {
    scanf("%d %d",&n,&N);
    N=1<<N;
    int sum=0;
    scanf("%d %d %d",&x,&y,&z);
    for(int i=0,a,b,c; i<n; i++) {
        scanf("%d %d %d",&a,&b,&c);
        sum^=a;
        b^=a;
        c^=a;
        a1[b]++;
        a2[c]++;
        a3[b^c]++;
    }
    xor_FWT(a1,1);
    xor_FWT(a2,1);
    xor_FWT(a3,1);
    for(int i=0; i<N; i++) {
        int c1=(1ll*n+a1[i]+a2[i]+a3[i])%MOD/4;
        int c2=(1ll*n+a1[i]-c1-c1+MOD)%MOD/2;
        int c3=(1ll*n+a2[i]-c1-c1+MOD)%MOD/2;
        int c4=(1ll*n+a3[i]-c1-c1+MOD)%MOD/2;
        ll ans=1;
}

```

```
ans=ans*quick_power((1ll*x+y+z),c1,MOD)%MOD;
ans=ans*quick_power((1ll*x+y-z+MOD)%MOD,c2,MOD)%MOD;
ans=ans*quick_power((1ll*x-y+z+MOD)%MOD,c3,MOD)%MOD;
ans=ans*quick_power((1ll*x-y-z+MOD*2)%MOD,c4,MOD)%MOD;
b1[i]=ans;
}
xor_FWT(b1,-1);
for(int i=0;i<N;i++) {
    printf("%lld ",b1[i^sum]);
}
return 0;
}
```

From:
<https://wiki.cvbbacm.com/> - CVBB ACM Team

Permanent link:
https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:%E7%8E%8B%E6%99%BA%E5%BD%AA:fwt&rev=1626850363

Last update: 2021/07/21 14:52

