FWT快速沃尔什变换

算法思想

或卷积

大概长这个样子 $C_{k} = \sum_{i|j=k} A_{i}×B_{j}$

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：

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

$FWT(A) = (FWT(A_{0}),FWT(A_{0})+FWT(A_{1}))$ 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 $FWT(A)[I] = \sum_{j|i=i}A[j]$

此时用数学归纳法有：

$FWT(A|B)$

$=FWT((A|B)_{0},(A|B)_{1})$

$=FWT(A_{0}|B_{0},A_{0}|B_{1}+A_{1}|B_{0}+A_{1}|B_{1})$

$=(FWT(A_{0}|B_{0}),FWT(A_{0}|B_{0}+A_{0}|B_{1}+A_{1}|B_{0}+A_{1}|B_{1}))$

$=(FWT(A_{0})×FWT(B_{0}),FWT(A_{0})×FWT(B_{0})+FWT(A_{0})×FWT(B_{1})+FWT(A_{1})×FWT(B_{0})+FWT(A_{1})×FWT(B_{1}))$

$=(FWT(A_{0})×FWT(B_{0}),(FWT(A_{0})+FWT(A_{1}))×(FWT(B_{0})+FWT(B_{1})))$

$=(FWT(A_{0},FWT(A_{0}+A_{1})))×(FWT(B_{0},FWT(B_{0}+B_{1})))$

$=FWT(A)×FWT(B)$

得证

所以类似于 FFT 也有一个叫做 $IFWT$ 的东西。

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

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 $n≤10^{5},N≤17$

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。

我们发现现在要求的是 $S[i]=\prod_{k=1}^{n} FWT(F_{k})[i]$ ，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 $O(logn)$ ，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 这个数异或，这样元组变为 $(0,b_{k}⊕a_{k},c_{k}⊕a_{k})$ ，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 $b_{k}⊕c_{k}$ 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 $O((k+logn)2^{k})$ ，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;
}
```

2.https://www.luogu.com.cn/problem/P6097

是一个模板题，又叫子集卷积，在原来的或卷积的基础 $i|j=k$ 上，加上了 $i\&j=0$ 的限制条件。

$i|j=k$ 的限制条件还是用 FWT 计算卷积。

对于第二个限制，等价于 $popcount(i)+popcount(j)=popcount(i|j)$ 首先显然要新开一维记录每个位置的 $popcount$ 然后让 $f_{i,j}=a_{j}(popcount(j)=i)$ 不然 $f_{i,j}=0$ $g_{i,j}$ 同理，然后有 $h_{i}=\sum_{k=0}^{i}f_{k}\times g_{i-k}$ 最后所求答案为 $h_{popcount(i),i}$ 这样复杂度是 $O(n^{2}2^{n})$ 的。

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int MAXN=1048577;
const ll MOD=1e9+9,inv2=(MOD+1)>>1;

int N,lim,pc[MAXN];
int a2[21][MAXN],b2[21][MAXN],ans[21][MAXN];

void or_FWT(int *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 solve_or_FWT() {
    for(int i=0;i<=lim;i++) or_FWT(a2[i],1),or_FWT(b2[i],1);
    for(int i=0;i<=lim;i++) {
        for(int j=0;j<=i;j++) {
            for(int k=0;k<N;k++) {
                ans[i][k]=(ans[i][k]+1ll*a2[j][k]*b2[i-j][k]%MOD)%MOD;
            }
        }
    }
```

```
    for(int i=0;i<=lim;i++) or_FWT(ans[i],-1);
}

void print_or_FWT() {
    for(int i=0;i<N;i++) printf("%d ",ans[pc[i]][i]);
    putchar(10);
}

int main() {
    scanf("%d",&N);
    lim=N;
    N=1<<N;
    for(int i=0;i<N;i++) pc[i]=__builtin_popcount(i);
    for(int i=0;i<N;i++) scanf("%d",&a2[pc[i]][i]);
    for(int i=0;i<N;i++) scanf("%d",&b2[pc[i]][i]);
    solve_or_FWT();
    print_or_FWT();
    return 0;
}
```

3.牛客第七场E

题意

给了 n 堆石子和 m 个可以取走的石子的数量，记为 x_{i} 除了这 m 种石子，还可以取莫比乌斯函数值为 1 的数量石子。同时给出这 n 堆石子的数量范围 $[l_{i},r_{i}]$ 求所有的情况中，先手必胜的局数，局面不同当且仅当存在一堆石子在两个局面中数量不同。

$1≤n≤10^{6},1≤l_{i},r_{i}≤10^{5},1≤m≤5,1≤x_{i}≤10^{5}$

题解

大综合题，显然需要会莫比乌斯反演（废话），还需要会博弈论的 sg 那套理论，先手必胜当且仅当所有的 sg 值异或起来不为 0。听出题人说 sg 值最高不会超过 230，但是现场的时候怎么知道嘛…

显然需要先筛出来 1 到 100000 的莫比乌斯函数值，才能知道哪些能加进去，然后再把 m 种数字也放进去，然后就是看每个状态的后继状态，这里也是学到了可以暴力搞。

对于每一个新的值 i 搞一个数组，看这个值的后继状态有没有 sg 为这个值的，如果有需要继续找，直到找不到，根据 mex 那套理论，就是此时的 sg 了。然后 i 加上刚才放进去的那些数的后继状态可以到达 i 于是这些状态的后继 sg 值可以有现在这个值，这里我们只关心能不能有，所以可以用 $bitset$ 优化一下，这部分的复杂度是 $O({\frac {100000^{2}} w}+256×100000)$ 这里本来要写 230 的，但是后面需要变成 256。

于是我们处理出了所有石子数的 sg 值，接下来对于每一堆石子，我们可以求出来，他们这个范围内，分别有多少个 sg 值为 0 的，有多少个 sg 值为 1 的，依次类推。

然后我们需要关心有多少个 $a[1]$ ^ $a[2]$ ^ $...$ ^ $a[n]≠0$ 这玩意显然可以看成 FWT 我们先统计出每个区间内有多少个 sg 值为 i 的，这显然前缀和就可以。然后我们相当于看 n 个多项式相

乘，每个多项式的幂次是各个 sg 值，系数是有多少个状态的 sg 值是这个值。如果对于每个求 FWT 再乘起来，最后再 $IFWT$ 回我们需要开到 256，这就照应了前面。算了一下，单个复杂度是 $O(256\times log_{2}(256))=O(2048)$ 然后 n 个就是 2×10^{9} 的，这显然是自杀行为。

然后 $oi-wiki$ 上一段话刷新了我对 FWT 的认知：若我们令 $i\&j$ 中 1 的奇偶性为 i 与 j 的奇偶性，于是 i 与 k 的奇偶性异或 j 与 k 的奇偶性等于 i ^ j 与 k 的奇偶性。然后可以得到异或的 FWT 是 $A_{i}=\sum_{C_{1}}A_{j}-\sum_{C_{2}}A_{j}$ 其 C_{1} 表示 $i\&j$ 的奇偶性为偶 C_{2} 表示 $i\&j$ 的奇偶性为奇。（其实记住就行…）

又因为 sg 的范围有限，所以我们可以先预处理出每个数的二进制有多少位为 1（或者直接 $\_\_builtin\_popcount$ 也可以）。

然后对于前缀和数组，就不能直接暴力加一了，判断两者与的奇偶性，如果为偶，则加一，不然减一。然后这样就直接把每一堆的 FWT 数组给求完了，复杂度变成 $O(256n)$ 而不是原来的 $O(2048n)$ 再全乘起来求一个 $IFWT$ 就可以了，整个这部分的复杂度都是 $O(256n)$ 的，最后对于所有 sg 值不为 0 的结果都加起来就是答案了。

另外这题卡常！前缀和数组必须大的做第一维，我不倒过来会 t 掉绝大多数数据，倒过来效率就第一了…

```
const int maxn=100000,N=1e6+10,maxm=256,MOD=1e9+7,inv2=500000004;
bool check[maxn+1];
int prime[maxn+1],mu[maxn+1];
int
tot,n,m,ls[N],rs[N],sg[maxn+1],sum[maxn+1][maxm+1],ans[maxm+1],o[maxm+1];
bitset<maxn+1> t,b[maxm];


void Moblus() {
    mu[1]=1;
    for(int i=2; i<=maxn; i++) {
        if(!check[i]) {
            mu[i]=-1;
            prime[tot++]=i;
        }
        for(int j=0; j<tot; j++) {
            if(i*prime[j]>maxn) break;
            check[i*prime[j]]=true;
            if(i%prime[j]==0) {
                mu[i*prime[j]]=0;
                break;
            } else {
                mu[i*prime[j]]=-mu[i];
            }
        }
    }
}

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

void init() {
    o[0]=0;
    for(int i=1;i<maxm;i++)o[i]=o[i>>1]+(i&1);
    for(int i=1; i<=maxn; i++) {
        if(mu[i]==1)t.set(i);
    }
    for(int i=0; i<=maxn; i++) {
        sg[i]=0;
        while(b[sg[i]][i]) sg[i]++;
        b[sg[i]]|=(t<<i);
    }
}

int main() {
    Moblus();
    read(n);read(m);
    for(int i=1; i<=n; i++) read(ls[i]),read(rs[i]);
    for(int i=1,tmp; i<=m; i++) read(tmp),t.set(tmp);
    init();
    for(int i=0; i<=maxn; i++)
        for(int j=0; j<maxm; j++)
            if(!(o[sg[i]&j]&1))sum[i][j]=1;
            else sum[i][j]=MOD-1;
    for(int i=1; i<=maxn; i++) {
        for(int j=0; j<maxm; j++) {
            sum[i][j]=sum[i][j]+sum[i-1][j];
            if(sum[i][j]>=MOD) sum[i][j]-=MOD;
        }
    }
    for(int i=0; i<maxm; i++)ans[i]=1;
    for(int i=1; i<=n; i++)
        for(int j=0; j<maxm; j++)
            ans[j]=(ll)ans[j]*(sum[rs[i]][j]-sum[ls[i]-1][j])%MOD;
    xor_FWT(ans,-1,maxm);
    ll sum=0;
    for(int i=1; i<maxm; i++) {
        sum=sum+ans[i];
        if(sum>=MOD) sum-=MOD;
    }
    printf("%lld\n",(sum+MOD)%MOD);
    return 0;
```

```
}
```