胡写的板子part2 —— 三倍Ice☆Creeeeeeeeeeeam！！！~

更厉害的数据结构

仍然字符串 —— SA后缀数组

- 和后缀自动机一起玩耍——
- 基数排序比较神奇，具体内容见板子1...
- 除了长度外**字符集大小**有影响，有个地方开桶的大小要考虑这个的！所以离散化~！

李超线段树

- 动态维护一个平面直角坐标系，可以在中间插入一条线段，回答询问在与x=x0这条直线相交的所有线段中，交点的y轴坐标的最大(小)值。
- 就像一个..凸.包..？（不准确）
- 开一棵线段树，标记下放
- 时间复杂度：
  修改部分：我们每次把一条线段的值域分隔到O(logn)个区间，每个区间最多把标记下传O(logn)层，因此修改的时间复杂度为O(log2n)。（但实测常数很小）
  查询部分：每次在线段树上从上到下扫一遍，时间复杂度为O(logn)

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<cmath>
using namespace std;
const int maxn=1e6+5;
/*inline void read(int &n){
    char c='+';int x=0;bool flag=0;
    while(c<'0'||c>'9'){c=getchar();if(c=='-')flag=1;}
    while(c>='0'&&c<='9'){x=x*10+c-'0';c=getchar();}
    n=flag==1?-x:x;
}*/
struct f{
    double k,b;
}a[maxn];//直线...
int tree[maxn];//线段树...
int tot=0;//线段树计数...

double ans=0;
bool pd(int now,int pre,int day){
    day--;
    return a[now].k*day+a[now].b>a[pre].k*day+a[pre].b;
}//day这一天对应的y值（y=k*x+b），第1天是b，第二天是1*k+b，所以day要减一

#define ls id<<1
#define rs id<<1|1

void insert(int id,int l,int r,int now){
```

```
    if(l==r){
        if(pd(now,tree[id],l))tree[id]=now;//如果叶子节点，当前直线在之前直线的“上边”就直接修改
        return;
    }
    int mid=(l+r)>>1;
    //下边都是下放，如果当前斜率更大，那么在交点左侧（负半轴方向）的直线是原来的直线在“上边”，反之同理
    //下放那个在当前不占优势的直线
    if(a[now].k>a[tree[id]].k){
        if((pd(now,tree[id],mid)))insert(ls,l,mid,tree[id]),tree[id]=now;
        else insert(rs,mid+1,r,now);
    }
    else {
        if((pd(now,tree[id],mid)))insert(rs,mid+1,r,tree[id]),tree[id]=now;
        else insert(ls,l,mid,now);
    }
}

void query(int id,int l,int r,int now){//now就是：直线里的x
    ans=max(ans,a[tree[id]].k*(now-1)+a[tree[id]].b);
    if(l==r)return;
    int mid=(l+r)>>1;
    if(now<=mid)query(ls,l,mid,now);//可能直线都来一遍，上边在上的不一定比下放的更优
    else query(rs,mid+1,r,now);
}

int main()
{
    ios::sync_with_stdio(false);
    int n;
    cin>>n;
    for(int i=1;i<=n;i++){
        string opt;
        cin>>opt;
        if(opt[0]=='P'){
            ++tot;
            cin>>a[tot].b>>a[tot].k;
            insert(1,1,n,tot);
        }
        else {
            int p;
            cin>>p;
            ans=0;
            query(1,1,n,p);
            printf("%d\n",(int)(ans/100));//那个题规定到百元才÷100..
        }
    }
    return 0;
}
```

主席树+动态主席树

主席树

- 别管名字多花里胡哨
- 能支持查询区间l~r的第k大！思想是前缀和！

- 考虑每次插入一个数就造一棵线段树，线段树上点不是存代表线段l~r区间信息，而是数值在那个l_num~r_num区间的**数字出现的个数**
  这里的数值要离散化哦！
- 如果这样的话，询问数组区间l~r就是两颗线段树（l-1和r两颗）对应点的前缀和做差会得到数值在l_num~r_num在区间l~r出现的次数
- 然后两边分开走就好啦（左边个数比k多走左边k=k，左边个数比k少走右边k=k-左个数），下边是板子
- 这里主要要解决问题是如果真的都分开开线段树会炸空间，解决方案是重用之前的点
  （多加一个数最多修改一条链logn个点，只要新开logn个点）

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
const int maxn=2e5+5;
struct node{
    int l,r,sum;
}T[maxn*20];//总每个树节点记录
int cnt=0;//节点数计数
int a[maxn];//原数组
int h[maxn];//离散化用
int rot[maxn];//第i个树的根记录

void build(int &rt,int l,int r){
    rt=++cnt;
    T[rt].sum=0;
    if(l==r)return;
    int mid=(l+r)>>1;
    build(T[rt].l,l,mid);
    build(T[rt].r,mid+1,r);
}//先造个第0棵树--

void update(int &rt,int prert,int l,int r,int num,int val){//这里写的就是rt从prert
继承的东西
    rt=++cnt;
    T[rt].sum=T[prert].sum+val;
    T[rt].l=T[prert].l;
    T[rt].r=T[prert].r;//记着左右子树也继承过来，如果改会在下边改的
    if(l==r)return;
    int mid=(l+r)>>1;//左改 or 右改
    if(num<=mid)update(T[rt].l,T[prert].l,l,mid,num,val);
    else update(T[rt].r,T[prert].r,mid+1,r,num,val);
}

int query(int rtl,int rtr,int l,int r,int k){//rtl，rtr传根；l，r就是值...
    if(l==r)return l;
    int cutcut=T[T[rtr].l].sum-T[T[rtl].l].sum;//判左值
    int mid=(l+r)>>1;
    if(cutcut>=k)return query(T[rtl].l,T[rtr].l,l,mid,k);
    else return query(T[rtl].r,T[rtr].r,mid+1,r,k-cutcut);//记着写k-cutcut
}

int main()
{
    int n,m;
```

```
    scanf("%d%d",&n,&m);

    for(int i=1;i<=n;i++)scanf("%d",&a[i]),h[i]=a[i];
    sort(h+1,h+1+n);
    int num=unique(h+1,h+1+n)-(h+1);//离散化

    build(rot[0],1,num);
    for(int i=1;i<=n;i++){
        a[i]=lower_bound(h+1,h+1+num,a[i])-h;//离散化
    //  cout<<a[i]<<endl;
        update(rot[i],rot[i-1],1,num,a[i],1);//更新造新树
    }

    for(int i=1;i<=m;i++){
        int l,r,k;
        scanf("%d%d%d",&l,&r,&k);
        printf("%d\n",h[query(rot[l-1],rot[r],1,num,k)]);//查询输出原数，query返回
序号，h[]还原原数字
    }

    return 0;
}
```

动态主席树

- 差不多可以叫树状数组套线段树了ovo，实际上已经和主席树略偏离了，继承前一个消失，前缀思想还在。
- 如果我们修改一个点（位置是pos）的话，就相当于第pos棵树**到**最后一棵树都统一修改那个数值所在链，暴力修改大概O(nlogn)是不行的
- 如果说修改是从（位置还是pos）数字a改变成b，就是pos到最后树都减a（修改链）然后加b（修改链），
- 那我们把这些操作放到树状数组里就是O(logn*logn)!
- 仅log个修改，同时与之对应，前缀和的O(1)查询就会变成O(logn)，也就是用了log棵树做差
- 不用想太复杂的，就是一个数组单点修改区间查询，只不过数组每个空存的是线段树，而这个数组又恰好是树状数组哦!
- 存一个数字修改就改个数字，那是因为他的修改方式就是改字。但是线段树，就需要按修改线段树的方法修改啦
- 看板子叭!

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
const int maxn=2e5+5;
struct node {
    int l,r,sum;
} T[maxn*400];//蜜汁400倍...小了就re，这计算...
int cnt=0;
int a[maxn];
int h[maxn];
int rot[maxn];

struct opts {
    int a,b,c;
    bool is_q;
```

```
} op[maxn];
char opop[5];
int num;
int many[2];
int log_go[2][30];

void update(int &rt,int l,int r,int pos,int val) {
    if(!rt)rt=++cnt;//判点
    T[rt].sum+=val;
    if(l==r)return;
    int mid=(l+r)>>1;
    if(pos<=mid)update(T[rt].l,l,mid,pos,val);
    else update(T[rt].r,mid+1,r,pos,val);
}//削弱继承机制，因为是树套树，现场造点

int query(int l,int r,int k){
    if(l==r)return l;
    int mid=(l+r)>>1;
    int sum=0;
    for(int i=1;i<=many[1];i++)sum+=T[T[log_go[1][i]].l].sum;
    for(int i=1;i<=many[0];i++)sum-=T[T[log_go[0][i]].l].sum;
    //上边是按log棵树计算sum，作比较，还是左儿子的sum和k，然后下边就是log棵树同时跳然后递归
    if(k<=sum){
        for(int i=1;i<=many[1];i++)log_go[1][i]=T[log_go[1][i]].l;
        for(int i=1;i<=many[0];i++)log_go[0][i]=T[log_go[0][i]].l;
        return query(l,mid,k);
    }
    else {
        for(int i=1;i<=many[1];i++)log_go[1][i]=T[log_go[1][i]].r;
        for(int i=1;i<=many[0];i++)log_go[0][i]=T[log_go[0][i]].r;
        return query(mid+1,r,k-sum);
    }
}

int c[maxn];
int lowbit(int x){return x&(-x);}

void log_add(int x,int val){//按树状数组来，因此继承机制退去
    int k=lower_bound(h+1,h+1+num,a[x])-h;//这里按a[x]写的，所以下边的修改要记得改a
    for(int i=x;i<=num;i+=lowbit(i))update(rot[i],1,num,k,val);//log更新
}

int log_query(int l,int r,int k){
    memset(log_go,0,sizeof(log_go));//感觉可以不写...亲测也可以不写...
    many[0]=many[1]=0;//赋值0个元素
    for(int i=r;i;i-=lowbit(i))log_go[1][++many[1]]=rot[i];//先加，后边都是1~many，
跳log个树
    for(int i=l-1;i;i-=lowbit(i))log_go[0][++many[0]]=rot[i];
    return query(1,num,k);
}

int main() {
    int n,m;
    scanf("%d%d",&n,&m);

    for(int i=1; i<=n; i++)scanf("%d",&a[i]),h[i]=a[i];
    num=n;
    for(int i=1; i<=m; i++) {
```

```
        scanf("%s",opop);
        if(opop[0]=='Q') {
            scanf("%d%d%d",&op[i].a,&op[i].b,&op[i].c);
            op[i].is_q=1;//判断是不是query操作
        }
        else {
            scanf("%d%d",&op[i].a,&op[i].b);
            op[i].is_q=0;
            h[++num]=op[i].b;//改的数也要离散化
        }
    }

    sort(h+1,h+1+num);
    num=unique(h+1,h+1+num)-(h+1);

    for(int i=1; i<=n; i++) log_add(i,1);
    for(int i=1;i<=m;i++){
        if(op[i].is_q)printf("%d\n",h[log_query(op[i].a,op[i].b,op[i].c)]);
        else {
            log_add(op[i].a,-1);//改-1，实际上就是直接暴力插点改值
            a[op[i].a]=op[i].b;//改a，为下一步搞事铺垫
            log_add(op[i].a,1);//改1，again
        }
    }
    return 0;
}
```

- 感觉就是名次树+前缀和+线段树=主席树

名次树

- 就是BST二叉平衡树上加个size表示子树大小（包括自己）
- 然后感觉就是主席树的感觉了
- Treap写！set搞不定了。。

BST

- O(logn)查询+删除+插入啦
- Treap/Splay
- 一般来讲set够用/红黑树
- 蓝书瞎写吧...代码有空补上...

更多数学

基尔霍夫矩阵/拉普拉斯矩阵 & 矩阵树定理

- L=D-W这是那个矩阵的一个求法~（度数矩阵（只有对角线不是0）-邻接矩阵）
  - (这里介绍下度数矩阵，度数矩阵只有对角线上不是0，第i行第i列的值是点i的度数，也就是邻接矩阵这一行的和)
  - (这里的邻接矩阵**允许重边**，邻接矩阵第i行第j列的值是i和j之间有几条边)
- 矩阵树定理解决一个很简单的问题，一个无向图到底有多少个生成树?
  - **1.Kirchhoff矩阵同时去掉任意一行一列，剩下的矩阵行列式绝对值相等**
  - **2.Kirchhoff矩阵同时去掉任意一行一列，剩下的矩阵的行列式绝对值，就是这个无向图的生成树个数**

    - 知乎上有场论+拉普拉斯搞过来的对图梯度求散度的解释...
- 大概来说就是：拉普拉斯矩阵是n*n的对角线是点i的度数（无向图），然后其他的表示邻接矩阵...
- 行列式计算就是削成上/下三角矩阵然后对角线相乘就是答案!
    - 生成树的多少就是直接高斯消元走啦...行列式变换好像要换符号的...
- 一个特例:完全图的生成树个数是n^{n-2}
- 代码补上（SHOI2016 黑暗前的幻想乡）

```
#include<iostream>
#include<cstdio>
using namespace std;
const int maxn=20;
const int mod=1e9+7;
using ll=long long;
int n;
int m[maxn],u[maxn][maxn*maxn],v[maxn][maxn*maxn];//存边...
int siz[300000];//容斥原理的size大小用... 2^maxn大小至少...

ll qp(ll a,ll p){
    ll res=1,base=a;
    while(p){
        if(p&1)res=res*base%mod;
        base=base*base%mod;
        p>>=1;
    }
    return res;
}

ll koff[maxn][maxn];//Kirchhoff矩阵!
inline ll det(){//行列式板板ovo
    ll res=1;
    int tr=0;//+，-反号次数
    for(int i=1;i<=n-1;i++){
        if(koff[i][i]==0){
            for(int j=i+1;j<=n-1;j++){
                if(koff[j][i]==0){continue;}
                tr^=1;
                for(int k=1;k<=n-1;k++){swap(koff[i][k],koff[j][k]);}
            }
        }//先找个[i][i]位有数的换到i行

        for(int j=i;j<=n-1;j++){
            if(koff[j][i]==0){continue;}//后边行的第i个是0就不用消除了
            ll div=qp(koff[j][i],mod-2);
            //模意义下的除变乘逆元/费马小定理，那么这一行和第i行的相减就是模意义下的0啦
            res=(res*koff[j][i])%mod;
            //对于一方阵其一行/一列乘以k，行列式乘k（这里的k就是这个koff）
            for(int k=i;k<=n-1;k++)koff[j][k]=(koff[j][k]*div)%mod;//当前行全体乘
div
        }

        for(int j=i+1;j<=n-1;j++){//做差消0
            if(koff[j][i]==0){continue;}
            for(int k=i;k<=n-1;k++){
                koff[j][k]=(koff[j][k]+mod-koff[i][k])%mod;
            }
        }
```

```
    }
    for(int i=1;i<=n-1;i++)res*=koff[i][i];
    return tr ? (mod-res)%mod : res;
}

int main()
{
    scanf("%d",&n);
    int upnum=(1<<(n-1))-1;
    for(int i=1;i<n;i++){
        scanf("%d",&m[i]);
        for(int j=1;j<=m[i];j++){
            scanf("%d%d",&u[i][j],&v[i][j]);
        }
    }

    for(int i=1;i<=upnum;i++)siz[i]=siz[i>>1]+(i&1);

    long long ans=0;
    for(int i=1;i<=upnum;i++){
        for(int j=1;j<=n;j++)
            for(int k=1;k<=n;k++)koff[j][k]=0;

        for(int j=1,p=i;p;p>>=1,j++){
            if((p&1)==0)continue;//p看有没有当前位...
            for(int k=1;k<=m[j];k++){
                int bg=u[j][k],ed=v[j][k];
                koff[bg][bg]++,koff[ed][ed]++;
                koff[bg][ed]=(koff[bg][ed]+mod-1)%mod;
                koff[ed][bg]=(koff[ed][bg]+mod-1)%mod;
                //-1是负的邻接矩阵
            }
        }

        ans=(ans+mod+((n-siz[i])%2 ? det():-det()))%mod;//看好括号...
    }
    printf("%lld\n",ans);
    return 0;
}
```

承接banzi1没有的数学——

莫比乌斯反演/狄利克雷卷积

- 终于我还是要看这个哈哈哈!

先来点欧拉函数

- 欧拉函数，符号记作φ(n)，其值为小于n且与n互质的数的个数
  - 性质:
  - 1.对于质数n：φ(n)=n−1
  - 2.对于
    $n = p^k, \phi(n) = (p-1) * p^{k-1}$
  - 3.积性函数，对于gcd(n,m)=1
    $\phi(n*m) = \phi(n) * \phi(m)$

- 4.计算式，对于 $n=\prod p_i^{k_i}, \phi(n)=n*\prod(1-\frac{1}{p_i})$
- 5.欧拉定理，对于互质的a，m有 $a^{\phi(m)} \equiv 1 \pmod{m}$
- 6.小于n且与n互质的数的和：$S=n*\frac{\phi(n)}{2}$
- 7.对于质数p，若：$n\ mod\ p=0, \phi(n*p)=\phi(n)*p$ $n\ mod\ p\neq 0, \phi(n*p)=\phi(n)*(p-1)$
- 8.$\sum_{d|n}\phi(d)=n, \phi(n)=\sum_{d|n}\mu(d)*\frac{n}{d}$

- 线性筛欧拉函数φ：

```
void getphi() {
    phi[1]=1;
    for(int i=2; i<=N; i++) {
        if(!vis[i])  {
            prime[++tot]=i;
            phi[i]=i-1;//欧拉函数素数直接i-1
        }
        for(int j=1; j<=tot; j++) {
            if(i*prime[j]>N)  break;
            vis[i*prime[j]]=1;
            if(i%prime[j]==0) {
                phi[i*prime[j]]=phi[i]*prime[j];//欧拉函数性质7
                break;
            } else phi[i*prime[j]]=phi[i]*(prime[j]-1);//欧拉函数性质7
        }
    }
}
```

莫比乌斯反演

就这玩意：

$$F(n)=\sum_{d|n}f(d)\Rightarrow f(n)=\sum_{d|n}\mu(d)F(\frac{n}{d}), f(n)=\sum_{d|n}\mu(\frac{n}{d})F(d)$$

- μ就是莫比乌斯函数啦，定义如下：
  - (1)若d=1则μ(d)=1;
  - (2)若 $d=p_1*p_2*...*p_k, p_i$为互异素数 那么μ(d)=(-1)^k;说直白点，就是d分解质因数后，没有幂次大于平方的质因子，此时函数值根据分解的个数决定
  - (3)其他情况下μ(d)=0，只要当d含有任何质因子的幂次大于等于2，则函数值为0
- 作用就是f不好算但是F好算（约数和/倍数和）也是好算就可以反演嘿嘿嘿
- 注意到这玩意那个**互异素数**的个数，这和容斥原理的奇偶挺相似的，所以一些题考虑一波容斥原理！

莫比乌斯函数的性质

- 1.对于任意正整数n$\sum_{d|n}\mu(d)=[n=1]$[n=1]表示只有当n=1成立时，返回值为1；否则，值为0。
  - 证明方法是二次项系数展开x=1,y=-1；这个就是说只有n=1这个和才是1，否则都是0。(这一条性质是莫比乌斯反演中最常用的)

    - 上边这个把n换成gcd(i,j);(ps:一般不都是gcd(i,j)=1？);就可以得到……
      $\sum_{d|gcd(i,j)}\mu(d)=[gcd(i,j)=1]$大多数直接带入...
- 2.对于任意正整数n$\sum_{d|n}\frac{\mu(d)}{d}=\frac{\phi(n)}{n}$
    - 非常神奇，μ和φ结合了...这就是为啥开头搞点欧拉函数
- 线性筛莫比乌斯函数μ：

```
void get_mu(int n)
{
    mu[1]=1;//在n=1的定义就是mu=1
    for(int i=2;i<=n;i++)
    {
        if(!vis[i]){prim[++cnt]=i;mu[i]=-1;}//奇数个素数mu=-1
        for(int j=1;j<=cnt&&prim[j]*i<=n;j++)
        {
            vis[prim[j]*i]=1;
            if(i%prim[j]==0)break;//这里阻挡了素数大于二次方的mu(prim[j]有多个)，其不改变均为mu=0
            else mu[i*prim[j]]=-mu[i];//因为多一个素数prim[j]，所以变号就行 i -> i*prim[j]
        }
    }
 }
```

实战前置数论技能——整除分块

- 考虑这个问题$\sum_{i=1}^{n}\lfloor\frac{n}{i}\rfloor$，n<1e12，O(n)是不可以的哦！(这经常出现)
- 整除分块出现O(√n)——
    - 经过一番冷静推理暴力打表，我们发现以下性质
    - 1.[N/i]最多只有2√N种取值
    - 2.设 [N/i'] 与 [N/i] 相等，则 i' 的最大值为 [N/[N/i]]
    - 所以：设两个指针 L 和 R， L 的初始值为 1，每次令 R=[N/[N/L]]，将 (R−L+1)*[N/L] 累加至答案中，再令 L=R+1，由于 [N/L] 只有 2√N 种取值，且单调递减，则最多只有 2√N 个取值不同的段，时间复杂度为 O(√N)

```
for(int l=1,r;l<=n;l=r+1)
{
    r=n/(n/l);
    ans+=(r-l+1)*(n/l);
}
```

假装实战

- 发现是自己数学公式推不来——累加...
- 求$\sum_{i=1}^{n}\sum_{j=1}^{n}[gcd(i,j)=1](n<m)$
- 直接上边带入$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{d|gcd(i,j)}\mu(d)$
- 然后要枚举d，变形公式(不会的就这个...)$=\sum_{d=1}^{n}\mu(d)*\lfloor\frac{n}{d}\rfloor*\lfloor\frac{m}{d}\rfloor$
- 然后上边的后半部分是整除分块
- 此题如果gcd(i,j)=k时，全体除以k变成：$\sum_{i=1}^{\lfloor\frac{n}{k}\rfloor}\sum_{j=1}^{\lfloor\frac{m}{k}\rfloor}[gcd(i,j)=1]$

杜教筛

- 讲道理这玩意是接着上边可以搞的

- 看了看实在太难了...
- 先不补了---
- 任务是小于O(n)筛$\sum_{i=1}^{n} f(n)$，f(n)是积性函数，就是f的前缀和

BM线性递推

- BM算法用于求解常系数线性递推式。
- 它可以在 O(n^2) 的时间复杂度内解决问题。
- 增量法什么的再补啦，先贴个板板

题目描述
给出一个数列P从0开始的前n项，求序列P在mod 998244353下的最短线性递推式，并在mod 998244353 下输出P_m

输入格式
第一行共两个数n,m,表示将会给出序列P的前n项，要求P_m
第二行n个数，表示P_0,P_1,P_2...P_{n-1}

输出格式
第一行输出该最短线性递推式。
第二行输出P_m的值。

```
//某谷多项式板子dalao全套，NTT都有了...以后再补自己的....
#include<bits/stdc++.h>
using namespace std;
typedef int ll;
typedef long long int li;
const ll MAXN=3e5+51,MOD=998244353,G=3,INVG=332748118;
ll exponent,fd,cnt=1,limit=-1,rres,ptr;
ll rev[MAXN],f[MAXN],g[MAXN],tmp[MAXN],tmp2[MAXN],tmp3[MAXN],tbm[MAXN];
ll res[MAXN],base[MAXN],fail[MAXN];
li delta[MAXN];
inline ll read()
{
    register ll num=0,neg=1;
    register char ch=getchar();
    while(!isdigit(ch)&&ch!='-')
    {
        ch=getchar();
    }
    if(ch=='-')
    {
        neg=-1;
        ch=getchar();
    }
    while(isdigit(ch))
    {
        num=(num<<3)+(num<<1)+(ch-'0');
        ch=getchar();
    }
    return num*neg;
}
inline ll qpow(ll base,ll exponent)
{
    li res=1;
    while(exponent)
    {
```

```
        if(exponent&1)
        {
            res=1ll*res*base%MOD;
        }
        base=1ll*base*base%MOD,exponent>>=1;
    }
    return res;
}
inline void NTT(ll *cp,ll cnt,ll inv)
{
    ll cur=0,res=0,omg=0;
    for(register int i=0;i<cnt;i++)
    {
        if(i<rev[i])
        {
            swap(cp[i],cp[rev[i]]);
        }
    }
    for(register int i=2;i<=cnt;i<<=1)
    {
        cur=i>>1,res=qpow(inv==1?G:INVG,(MOD-1)/i);
        for(register ll *p=cp;p!=cp+cnt;p+=i)
        {
            omg=1;
            for(register int j=0;j<cur;j++)
            {
                ll t=1ll*omg*p[j+cur]%MOD,t2=p[j];
                p[j+cur]=(t2-t+MOD)%MOD,p[j]=(t2+t)%MOD;
                omg=1ll*omg*res%MOD;
            }
        }
    }
    if(inv==-1)
    {
        ll invl=qpow(cnt,MOD-2);
        for(register int i=0;i<=cnt;i++)
        {
            cp[i]=1ll*cp[i]*invl%MOD;
        }
    }
}
inline void inv(ll fd,ll *f,ll *res)
{
    static ll tmp[MAXN];
    if(fd==1)
    {
        res[0]=qpow(f[0],MOD-2);
        return;
    }
    inv((fd+1)>>1,f,res);
    ll cnt=1,limit=-1;
    while(cnt<(fd<<1))
    {
        cnt<<=1,limit++;
    }
    for(register int i=0;i<cnt;i++)
    {
        tmp[i]=i<fd?f[i]:0;
```

```
        rev[i]=(rev[i>>1]>>1)|((i&1)<<limit);
    }
    NTT(tmp,cnt,1),NTT(res,cnt,1);
    for(register int i=0;i<cnt;i++)
    {
        res[i]=1ll*(2-1ll*tmp[i]*res[i]%MOD+MOD)%MOD*res[i]%MOD;
    }
    NTT(res,cnt,-1);
    for(register int i=fd;i<cnt;i++)
    {
        res[i]=0;
    }
}
inline void mod(ll *f)
{
    static ll tmp[MAXN],q[MAXN];
    ll deg=fd<<1;
    while(!f[--deg]);
    if(deg<fd)
    {
    return;
    }
    for(register int i=0;i<cnt;i++)
    {
        tmp[i]=i<=deg?f[i]:0;
    }
    reverse(tmp,tmp+1+deg);
    for(register int i=deg+1-fd;i<=deg;i++)
    {
        tmp[i]=0;
    }
    NTT(tmp,cnt,1);
    for(register int i=0;i<cnt;i++)
    {
        q[i]=(li)tmp[i]*tmp3[i]%MOD;
    }
    NTT(q,cnt,-1);
    for(register int i=0;i<cnt;i++)
    {
        tmp[i]=0;
        q[i]=i<=deg-fd?q[i]:0;
    }
    reverse(q,q+1+deg-fd),NTT(q,cnt,1);
    for(register int i=0;i<cnt;i++)
    {
        tmp[i]=(li)q[i]*g[i]%MOD;
    }
    NTT(tmp,cnt,-1);
    for(register int i=0;i<fd;i++)
    {
        f[i]=(f[i]-tmp[i]+MOD)%MOD;
    }
    for(register int i=0;i<cnt;i++)
    {
        q[i]=tmp[i]=0,f[i]=i<fd?f[i]:0;
    }
}
vector<li>bmf[MAXN];
```

```
inline void BerlekampMassey(ll length,ll *base,ll *res)
{
    ll cur=0;
    for(register int i=1;i<=length;i++)
    {
        li curr=base[i];
        for(register int j=0;j<bmf[cur].size();j++)
        {
            curr=(curr-(li)base[i-j-1]*bmf[cur][j]%MOD)%MOD;
        }
        delta[i]=curr;
        if(!delta[i])
        {
            continue;
        }
        fail[cur]=i;
        if(!cur)
        {
            bmf[++cur].resize(i),delta[i]=base[i];
            continue;
        }
        ll id=cur-1,x=bmf[id].size()-fail[id]+i;
        for(register int j=0;j<cur;j++)
        {
            if(i-fail[j]+bmf[j].size()<x)
            {
                id=j,x=i-fail[j]+bmf[j].size();
            }
        }
        bmf[cur+1]=bmf[cur],cur++;
        while(bmf[cur].size()<x)
        {
            bmf[cur].push_back(0);
        }
        li mul=(li)delta[i]*qpow(delta[fail[id]],MOD-2)%MOD;
        bmf[cur][i-fail[id]-1]=(li)(bmf[cur][i-fail[id]-1]+mul)%MOD;
        for(register int j=0;j<bmf[id].size();j++)
        {
            ll t=(li)mul*bmf[id][j]%MOD;
            bmf[cur][i-fail[id]+j]=(bmf[cur][i-fail[id]+j]-t+MOD)%MOD;
        }
    }
    ptr=cur;
    for(register int i=0;i<bmf[cur].size();i++)
    {
        res[i+1]=(bmf[cur][i]%MOD+MOD)%MOD;
    }
}
int main()
{
    fd=read(),exponent=read();
    for(register int i=0;i<fd;i++)
    {
        tbm[i+1]=f[i]=(read()+MOD)%MOD;
    }
    BerlekampMassey(fd,tbm,tmp);
    for(register int i=1;i<=bmf[ptr].size();i++)
    {
```

```
        printf("%d ",tmp[i]);
    }
    puts("");
    for(register int i=1;i<=fd;i++)
    {
        g[fd-i]=MOD-tmp[i];
    }
    g[fd]=1;
    for(register int i=0;i<=fd;i++)
    {
        tmp2[i]=g[i];
    }
    reverse(tmp2,tmp2+1+fd),inv(fd<<1,tmp2,tmp3);
    for(register int i=0;i<=fd;i++)
    {
        tmp2[i]=0;
    }
    while(cnt<(fd<<2))
    {
        cnt<<=1,limit++;
    }
    for(register int i=0;i<cnt;i++)
    {
        rev[i]=(rev[i>>1]>>1)|((i&1)<<limit);
    }
    NTT(g,cnt,1),NTT(tmp3,cnt,1),base[1]=res[0]=1;
    while(exponent)
    {
        if(exponent&1)
        {
            NTT(res,cnt,1),NTT(base,cnt,1);
            for(register int i=0;i<cnt;i++)
            {
                res[i]=(li)res[i]*base[i]%MOD;
            }
            NTT(res,cnt,-1),NTT(base,cnt,-1),mod(res);
        }
        NTT(base,cnt,1);
        for(register int i=0;i<cnt;i++)
        {
            base[i]=(li)base[i]*base[i]%MOD;
        }
        NTT(base,cnt,-1),mod(base),exponent>>=1;
    }
    for(register int i=0;i<fd;i++)
    {
        rres=(rres+(li)res[i]*f[i]%MOD)%MOD;
    }
    printf("%d\n",rres);
}
```

MTT & 分治FFT

FWT

二次剩余

Lucas定理 & EXLucas