算法总结补充：

BigInt:

Big Int

const int maxn=10005;

struct BigInt{

int len;//数字的长度

ll num[maxn];

void init(const string&s)//用string初始化BigInt

{

memset(num,0,sizeof(num));

len=s.length();

for(int i=1;i<=len;i++)num[i]=s[len-i]-'0';

}

void **init**(**ll** a)

    {

**memset**(num,0,sizeof(num));

        len=0;

        if(!a)num[++len]=0;

        else

        while(a){

            num[++len]=a%10;

            a/=10;

        }

    }

void print()//打印BigInt

{

for(int i=len;i>=1;i--)printf("%lld",num[i]);

}

friend BigInt operator +(const BigInt&a,const BigInt&b)

{

BigInt c;

c.init("");//c一开始是一个空串

int max\_len=max(a.len,b.len);

c.len=max\_len;

for(int i=1;i<=max\_len;i++)c.num[i]=a.num[i]+b.num[i];

//进位

ll tmp=0;

int pos=1;

while(1){

c.num[pos]+=tmp;

tmp=c.num[pos]/10;

c.num[pos]%=10;

pos++;

if(pos>max\_len)break;

}

while(tmp){

c.num[++c.len]=tmp%10;

tmp/=10;

}

while(1){

if(c.num[c.len]==0){

if(c.len==1)break;

c.len--;

continue;

}

break;

}

return c;

}

friend BigInt operator -(const BigInt&a,const BigInt&b)

{

BigInt c;

c.init("");

int max\_len=max(a.len,b.len);

for(int i=1;i<=max\_len;i++)c.num[i]=a.num[i]-b.num[i];

c.len=max\_len;

ll tmp=0;

int pos=1;

while(1){

c.num[pos]+=tmp;

if(c.num[pos]<0)tmp=-1,c.num[pos]+=10;

else tmp=0;

pos++;

if(pos>max\_len)break;

}

if(c.num[c.len]<0)c.num[c.len]+=10,c.num[++c.len]-=1;

while(1){

if(c.num[c.len]==0){

if(c.len==1)break;

c.len--;

continue;

}

break;

}

return c;

}

friend BigInt operator \*(const BigInt&a,const BigInt&b)

{

BigInt c;

c.init("");

c.len=a.len+b.len-1;

for(int i=1;i<=a.len;i++)

for(int j=1;j<=b.len;j++)

c.num[i+j-1]+=a.num[i]\*b.num[j];

ll tmp=0,pos=1;

while(1){

c.num[pos]+=tmp;

tmp=c.num[pos]/10;

c.num[pos]%=10;

pos++;

if(pos>c.len)break;

}

while(tmp){

c.num[++c.len]=tmp%10;

tmp/=10;

}

while(1){

if(c.num[c.len]==0){

if(c.len==1)break;

c.len--;

}

break;

}

return c;

}

friend BigInt operator /(const BigInt&a,const ll b)//高精度除longlong

{

BigInt c;

c.init("");

c.len=a.len;

ll tmp=0;

for(int i=1;i<=c.len;i++)c.num[i]=a.num[i];

for(int i=c.len;i>=1;i--){

c.num[i]+=tmp\*10;

tmp=c.num[i]%b;

c.num[i]/=b;

}

while(1){

if(c.num[c.len]==0){

if(c.len==1)break;

c.len--;

continue;

}

break;

}

return c;

}

};

2.线段树：

#include<bits/stdc++.h>

using namespace std;

const int maxn=(int)1e5+5;

long long a[maxn];

long long sum[4\*maxn],add\_tag[4\*maxn],mul\_tag[4\*maxn];//创建线段树，维护区间前缀和

int n,m,p;

void push\_up(int rt)

{

sum[rt]=(sum[2\*rt]+sum[2\*rt+1])%p;

return ;

}

void build(int ll,int rr,int rt)//建树

{

if(ll==rr){

sum[rt]=a[ll];

return ;

}

int mid=(ll+rr)/2;

//递归建左右子树

build(ll,mid,2\*rt);

build(mid+1,rr,2\*rt+1);

push\_up(rt);//回溯的过程向上更新子节点

}

void push\_down(int rt,int ln,int rn)

{

//默认在下移tag时先乘再加

sum[2\*rt]=(sum[2\*rt]\*mul\_tag[rt]+add\_tag[rt]\*ln)%p;

sum[2\*rt+1]=(sum[2\*rt+1]\*mul\_tag[rt]+add\_tag[rt]\*rn)%p;

//更新子节点的tag

mul\_tag[2\*rt]=(mul\_tag[2\*rt]\*mul\_tag[rt])%p;

mul\_tag[2\*rt+1]=(mul\_tag[2\*rt+1]\*mul\_tag[rt])%p;

add\_tag[2\*rt]=(add\_tag[2\*rt]\*mul\_tag[rt]+add\_tag[rt])%p;

add\_tag[2\*rt+1]=(add\_tag[2\*rt+1]\*mul\_tag[rt]+add\_tag[rt])%p;

//重置当前节点的tag，因为所有影响都已经下移

add\_tag[rt]=0;

mul\_tag[rt]=1;

}

void Mul(int L,int R,int ll,int rr,int rt,int k)

{

if(ll>=L&&rr<=R){

//现在的区间被目标区间完全包括

sum[rt]=(sum[rt]\*k)%p;

mul\_tag[rt]=(mul\_tag[rt]\*k)%p;

add\_tag[rt]=(add\_tag[rt]\*k)%p;

return;

}

int mid=(ll+rr)/2;

push\_down(rt,mid-ll+1,rr-mid);

if(mid>=L)Mul(L,R,ll,mid,2\*rt,k);

if(mid<R)Mul(L,R,mid+1,rr,2\*rt+1,k);

push\_up(rt);

}

void Add(int L,int R,int ll,int rr,int rt,int k)

{

if(ll>=L&&rr<=R){

//现在的区间被目标区间完全包括

sum[rt]+=k\*(rr-ll+1);

sum[rt]%=p;

add\_tag[rt]+=k;

add\_tag[rt]%=p;

return;

}

int mid=(ll+rr)/2;

push\_down(rt,mid-ll+1,rr-mid);

if(mid>=L)Add(L,R,ll,mid,2\*rt,k);

if(mid<R)Add(L,R,mid+1,rr,2\*rt+1,k);

push\_up(rt);

}

long long query(int L,int R,int ll,int rr,int rt)

{

if(ll>=L&&rr<=R){

return sum[rt];

}

int mid=(ll+rr)/2;

push\_down(rt,mid-ll+1,rr-mid);

long long ans=0;

if(L<=mid)ans+=query(L,R,ll,mid,2\*rt)%p;

if(mid<R)ans+=query(L,R,mid+1,rr,2\*rt+1)%p;

ans%=p;

return ans;

}

int main()

{

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

for(int i=1;i<=n;i++)

scanf("%lld",&a[i]);

build(1,n,1);

for(int i=1;i<=4\*n;i++){

mul\_tag[i]=1;

}//初始化

for(int i=1;i<=m;i++){

int opt,x,y,k;

scanf("%d",&opt);

if(opt==1){

//区间乘操作

scanf("%d %d %d",&x,&y,&k);

Mul(x,y,1,n,1,k);

}

else if(opt==2){

//区间加操作

scanf("%d %d %d",&x,&y,&k);

Add(x,y,1,n,1,k);

}

else{

scanf("%d %d",&x,&y);

long long ans=query(x,y,1,n,1);

printf("%lld\n",ans);

}

}

return 0;

}