// 多项式全家桶

namespace poly

{

struct Complex

{

double r,i;

Complex() {}

inline Complex(double curr,double curi)

{

r=curr;i=curi;

}

inline Complex operator + (Complex x)

{

return Complex(r+x.r,i+x.i);

}

inline Complex operator - (Complex x)

{

return Complex(r-x.r,i-x.i);

}

inline Complex operator \* (Complex x)

{

return Complex(r\*x.r-i\*x.i,r\*x.i+i\*x.r);

}

inline Complex operator / (double x)

{

return Complex(r/x,i/x);

}

}a[800048],b[800048];

int n,m;

//fl=1表示DFT,fl=-1表示iDFT

void FFT(Complex c[],int len,int fl)

{

int i,j,k,clen;

for (i=(len>>1),j=1;j<len;j++)

{

if (i<j) swap(c[i],c[j]);

for (k=(len>>1);i&k;k>>=1) i^=k;

i^=k;

}

for (clen=2;clen<=len;clen<<=1)

{

Complex wn=Complex(cos(pi\*2.0\*fl/clen),sin(pi\*2.0\*fl/clen));

//wn是clen次单位复根

for (j=0;j<len;j+=clen)

{

Complex w=Complex(1,0);

for (k=j;k<j+(clen>>1);k++)

{

Complex tmp1=c[k],tmp2=c[k+(clen>>1)]\*w;

c[k]=tmp1+tmp2;c[k+(clen>>1)]=tmp1-tmp2;

w=w\*wn;

}

}

}

if (fl==-1)

for (i=0;i<len;i++) c[i]=c[i]/len;

}

const int NTT\_MAX=8e5;

const int G=3;

int wn\_pos[NTT\_MAX+48],wn\_neg[NTT\_MAX+48];

inline int add(int x) {if (x>=MOD) x-=MOD;return x;}

inline int sub(int x) {if (x<0) x+=MOD;return x;}

inline int quick\_pow(int x,int y)

{

int res=1;

while (y)

{

if (y&1) res=(1ll\*res\*x)%MOD,y--;

x=(1ll\*x\*x)%MOD;y>>=1;

}

return res;

}

inline void poly\_init()

{

for (register int clen=2;clen<=262144\*2;clen<<=1)

wn\_pos[clen]=quick\_pow(G,(MOD-1)/clen),wn\_neg[clen]=quick\_pow(G,(MOD-1)-(MOD-1)/clen);

}

inline void poly\_ntt(int c[],int len,int fl)

{

int clen,i,j,k,w,wn,tmp1,tmp2;

for (i=(len>>1),j=1;j<len;j++)

{

if (i<j) swap(c[i],c[j]);

for (k=(len>>1);i&k;k>>=1) i^=k;i^=k;

}

for (clen=2;clen<=len;clen<<=1)

{

wn=(fl==1?wn\_pos[clen]:wn\_neg[clen]);

for (j=0;j<len;j+=clen)

{

w=1;

for (k=j;k<j+(clen>>1);k++)

{

tmp1=c[k];tmp2=(1ll\*c[k+(clen>>1)]\*w)%MOD;

c[k]=add(tmp1+tmp2);c[k+(clen>>1)]=sub(tmp1-tmp2);

w=(1ll\*w\*wn)%MOD;

}

}

}

if (fl==-1)

{

int inv=quick\_pow(len,MOD-2);

for (register int i=0;i<len;i++) c[i]=(1ll\*c[i]\*inv)%MOD;

}

}

inline void poly\_mul(int A[],int B[],int N,int M)

{

int len=1;while (len<=N+M) len<<=1;

memset(A+N,0,(len-N)\*sizeof(int));memset(B+M,0,(len-M)\*sizeof(int));

poly\_ntt(A,len,1);poly\_ntt(B,len,1);

for (register int i=0;i<len;i++) A[i]=(1ll\*A[i]\*B[i])%MOD;

poly\_ntt(A,len,-1);

}

inline void poly\_inv(int A[],int B[],int N)

{

B[0]=quick\_pow(A[0],MOD-2);

static int tmp[NTT\_MAX+48];int Len,i;

for (register int clen=2;(clen>>1)<N;clen<<=1)

{

Len=min(N,clen);

memcpy(tmp,A,Len\*sizeof(int));

memset(tmp+Len,0,((clen<<1)-Len)\*sizeof(int));

memset(B+(clen>>1),0,((clen<<1)-(clen>>1))\*sizeof(int));

poly\_ntt(tmp,clen<<1,1);poly\_ntt(B,clen<<1,1);

for (int i=0;i<(clen<<1);i++) B[i]=(1ll\*B[i]\*sub(2-(1ll\*tmp[i]\*B[i])%MOD))%MOD;

poly\_ntt(B,clen<<1,-1);

}

}

inline void poly\_div(int A[],int B[],int C[],int R[],int N,int M)

{

static int ar[NTT\_MAX+48],br[NTT\_MAX+48],tmp[NTT\_MAX+48];

int t=N-M+1,len=1;while (len<=(t<<1)) len<<=1;

memset(tmp,0,len\*sizeof(int));

reverse\_copy(B,B+M,tmp);

poly\_inv(tmp,br,len);

reverse\_copy(A,A+N,ar);

poly\_mul(ar,br,t,t);

reverse(ar,ar+t);memcpy(C,ar,t\*sizeof(int));

memset(br,B,M\*sizeof(int));

poly\_mul(ar,br,t,M);

for (register int i=0;i<M;i++) R[i]=sub(A[i]-ar[i]);

}

inline int calc\_sqrt(int x) {for (register int i=1;i<=MOD;i++) if ((1ll\*i\*i)%MOD==x) return i;}

inline void poly\_sqrt(int A[],int B[],int N)

{

B[0]=calc\_sqrt(A[0]);

static int tmp[NTT\_MAX+48],tinv[NTT\_MAX+48];int i,Len;

for (register int clen=2;(clen>>1)<N;clen<<=1)

{

Len=min(N,clen);

memcpy(tmp,A,Len\*sizeof(int));

memset(tmp+Len,0,((clen<<1)-Len)\*sizeof(int));

memset(B+(clen>>1),0,((clen<<1)-(clen>>1))\*sizeof(int));

poly\_inv(B,tinv,clen);

memset(tinv+clen,0,((clen<<1)-clen)\*sizeof(int));

poly\_ntt(tmp,clen<<1,1);poly\_ntt(B,clen<<1,1);poly\_ntt(tinv,clen<<1,1);

for (i=0;i<(clen<<1);i++)

B[i]=(1ll\*add(B[i]+(1ll\*tmp[i]\*tinv[i])%MOD)\*(MOD+1)/2)%MOD;

poly\_ntt(B,clen<<1,-1);

memset(B+clen,0,((clen<<1)-clen)\*sizeof(int));

}

}

}

// djq\_cpp的多项式模板

typedef long long LL;

typedef pair<int, int> PII;

const int MOD = 998244353;

inline int power(int x, int t)

{

int ret = 1;

while(t) {

if(t & 1) ret = LL(ret) \* x % MOD;

x = LL(x) \* x % MOD;

t >>= 1;

}

return ret;

}

//basic DFT

#define FIX\_DEG

const int MAXL = 20; //size: at least 4 times

const int UNR = 3;

int RT[2][MAXL + 1];

int IL[MAXL + 1];

int invs[(1 << MAXL) + 1];

int iret[(1 << MAXL) + 1];

inline void gen\_inv(int dat[], int n)

{

iret[0] = 1;

rep(i, n) iret[i + 1] = LL(iret[i]) \* dat[i] % MOD;

iret[n] = power(iret[n], MOD - 2) % MOD;

for(int i = n - 1; i >= 0; i --) {

int cur = iret[i];

iret[i] = LL(iret[i + 1]) \* dat[i] % MOD;

iret[i + 1] = LL(iret[i + 1]) \* cur % MOD;

}

rep(i, n) dat[i] = iret[i + 1];

}

inline int inv(int x)

{

if(x <= (1 << MAXL)) return invs[x];

return power(x, MOD - 2);

}

void fft\_init()

{

invs[0] = 1;

rep1(i, 1 << MAXL) invs[i] = LL(invs[i - 1]) \* i % MOD;

invs[1 << MAXL] = power(invs[1 << MAXL], MOD - 2) % MOD;

for(int i = (1 << MAXL) - 1; i >= 0; i --) {

int cur = invs[i];

invs[i] = LL(invs[i + 1]) \* (i + 1) % MOD;

invs[i + 1] = LL(invs[i + 1]) \* cur % MOD;

}

for(int i = 0; i <= MAXL; i ++) {

RT[0][i] = power(UNR, (MOD - 1) >> i);

RT[1][i] = inv(RT[0][i]);

}

for(int i = 0; i <= MAXL; i ++) IL[i] = inv(1 << i);

}

LL rts[1 << MAXL];

void DFT(int fr[], int to[], int len, bool fli)

{

int cur = 0;

rep(i, 1 << len) {

to[i] = fr[cur];

for(int j = len - 1; j >= 0; j --) {

cur ^= 1 << j;

if((cur >> j) & 1) break;

}

}

rep1(d, len) {

int cr = RT[fli][d];

LL cl = 1 << d - 1;

rep(i, cl) rts[i] = i == 0 ? 1 : rts[i - 1] \* cr % MOD;

for(int i = 0; i < (1 << len); i += cl << 1)

for(int j = i; j < i + cl; j ++) {

int ev = to[j], od = to[j + cl] \* rts[j - i] % MOD;

to[j] = ev + od;

if(to[j] >= MOD) to[j] -= MOD;

to[j + cl] = ev - od + MOD;

if(to[j + cl] >= MOD) to[j + cl] -= MOD;

}

}

if(fli) rep(i, 1 << len) to[i] = LL(to[i]) \* IL[len] % MOD;

}

//polynomial operations

struct poly

{

int\* beg;

int deg;

poly()

{

beg = NULL;

deg = -1;

}

poly(int cd)

{

beg = new int[cd + 1];

memset(beg, 0, cd + 1 << 2);

deg = cd;

}

~poly()

{

// if(beg != NULL) delete[] beg;

}

int& operator[](int id)

{

return beg[id];

}

};

void DFT(poly u, poly v, int len, bool fli)

{

DFT(u.beg, v.beg, len, fli);

}

void poly\_const(poly& ret, int dat)

{

ret = poly(0);

ret[0] = dat;

}

void poly\_line(poly& ret, int dat)

{

ret = poly(1);

ret[0] = dat == 0 ? 0 : MOD - dat;

ret[1] = 1;

}

void poly\_copy(poly& ret, poly u, int td)

{

ret = poly(td);

for(int i = 0; i <= min(ret.deg, u.deg); i ++) ret[i] = u[i];

#ifdef FIX\_DEG

while(ret.deg > 0 && ret[ret.deg] == 0) ret.deg --;

#endif

}

void poly\_add(poly& ret, poly u, poly v)

{

ret = poly(max(u.deg, v.deg));

for(int i = 0; i <= u.deg; i ++) ret[i] = u[i];

for(int i = 0; i <= v.deg; i ++) ret[i] = ret[i] + v[i];

for(int i = 0; i <= ret.deg; i ++) if(ret[i] >= MOD) ret[i] -= MOD;

#ifdef FIX\_DEG

while(ret.deg > 0 && ret[ret.deg] == 0) ret.deg --;

#endif

}

void poly\_sub(poly& ret, poly u, poly v)

{

ret = poly(max(u.deg, v.deg));

for(int i = 0; i <= u.deg; i ++) ret[i] = u[i];

for(int i = 0; i <= v.deg; i ++) ret[i] = ret[i] - v[i] + MOD;

for(int i = 0; i <= ret.deg; i ++) if(ret[i] >= MOD) ret[i] -= MOD;

#ifdef FIX\_DEG

while(ret.deg > 0 && ret[ret.deg] == 0) ret.deg --;

#endif

}

void poly\_smul(poly& ret, poly u, int v)

{

if(v == 0) {

ret = poly(0);

return;

}

ret = poly(u.deg);

for(int i = 0; i <= ret.deg; i ++) ret[i] = LL(u[i]) \* v % MOD;

}

void poly\_lmul(poly& ret, poly u, int v)

{

ret = poly(u.deg + 1);

for(int i = 0; i <= u.deg; i ++) {

ret[i] = (ret[i] + LL(u[i]) \* (MOD - v)) % MOD;

ret[i + 1] = u[i];

}

}

poly tr(1 << MAXL + 1);

poly tu(1 << MAXL + 1);

poly tv(1 << MAXL + 1);

void poly\_mul(poly& ret, poly u, poly v)

{

#ifdef FIX\_DEG

if(u.deg == 0 && u.beg[0] == 0 || v.deg == 0 && v.beg[0] == 0) {

ret = poly(0);

return;

}

#endif

ret = poly(u.deg + v.deg);

int cl = ret.deg == 0 ? 0 : 32 - \_\_builtin\_clz(ret.deg);

memset(tu.beg, 0, sizeof(int) << cl);

memset(tv.beg, 0, sizeof(int) << cl);

for(int i = 0; i <= u.deg; i ++) tu[i] = u[i];

for(int i = 0; i <= v.deg; i ++) tv[i] = v[i];

DFT(tu, tr, cl, false);

DFT(tv, tu, cl, false);

rep(i, 1 << cl) tv[i] = LL(tr[i]) \* tu[i] % MOD;

DFT(tv, tr, cl, true);

for(int i = 0; i <= ret.deg; i ++) ret[i] = tr[i];

}

void poly\_inv(poly& ret, poly u, int d\_n = -1)

{

assert(u[0] != 0);

if(d\_n == -1) d\_n = u.deg;

tr[0] = inv(u[0]);

for(int i = 0; (1 << i) <= d\_n; i ++) {

memset(tu.beg, 0, sizeof(int) << i + 2);

rep(j, 1 << i) tu[j] = tr[j];

memset(tv.beg, 0, sizeof(int) << i + 2);

rep(j, 1 << i + 1) tv[j] = j <= u.deg ? u[j] : 0;

DFT(tu, tr, i + 2, false);

DFT(tv, tu, i + 2, false);

rep(j, 1 << i + 2) tv[j] = (2 - LL(tu[j]) \* tr[j] % MOD + MOD) \* tr[j] % MOD;

DFT(tv, tr, i + 2, true);

}

int rd = d\_n;

#ifdef FIX\_DEG

while(rd > 0 && tr[rd] == 0) rd --;

#endif

ret = poly(rd);

for(int i = 0; i <= rd; i ++) ret[i] = tr[i];

}

//the functions below involves the functions above

#ifdef FIX\_DEG

void poly\_div(poly& ret, poly u, poly v)

{

if(u.deg < v.deg) {

ret = poly(0);

return;

}

poly ttu(u.deg - v.deg);

for(int i = 0; i <= ttu.deg; i ++) ttu[i] = u[u.deg - i];

poly ttv(u.deg - v.deg);

for(int i = 0; i <= ttv.deg; i ++) ttv[i] = i <= v.deg ? v[v.deg - i] : 0;

poly\_inv(ret, ttv);

poly\_mul(ttv, ret, ttu);

ret = poly(u.deg - v.deg);

for(int i = 0; i <= ret.deg; i ++) ret[i] = ttv[ret.deg - i];

}

void poly\_mod(poly& ret, poly u, poly v)

{

poly ttq, tts;

poly\_div(ttq, u, v);

poly\_mul(tts, v, ttq);

poly\_sub(ret, u, tts);

}

#endif

void poly\_diff(poly& ret, poly u)

{

if(u.deg == 0) {

ret = 0;

return;

}

ret = poly(u.deg - 1);

for(int i = 0; i <= ret.deg; i ++) ret[i] = LL(u[i + 1]) \* (i + 1) % MOD;

}

void poly\_int(poly& ret, poly u)

{

ret = poly(u.deg + 1);

for(int i = 0; i <= u.deg; i ++) ret[i + 1] = LL(u[i]) \* inv(i + 1) % MOD;

}

void poly\_ln(poly& ret, poly u, int d\_n)

{

assert(u[0] == 1);

poly v, w;

poly\_copy(w, u, d\_n);

poly\_diff(v, w);

poly\_inv(ret, w, d\_n);

poly\_mul(w, v, ret);

poly\_int(v, w);

poly\_copy(ret, v, d\_n);

}

void poly\_exp(poly& ret, poly u, int d\_n)

{

assert(u[0] == 0);

ret = poly(0);

ret[0] = 1;

poly v, w, x;

for(int i = 0; (1 << i) <= d\_n; i ++) {

poly\_copy(w, u, 1 << i + 1);

poly\_ln(v, ret, 1 << i + 1);

poly\_sub(x, w, v);

poly\_mul(v, ret, x);

poly\_add(w, v, ret);

poly\_copy(ret, w, min(1 << i + 1, d\_n));

}

}

poly mret[1 << MAXL + 1], msum[1 << MAXL + 1];

void poly\_prod(poly frm[], int n, int pos = 1)

{

if(n == 1) {

poly\_copy(mret[pos], frm[0], frm[0].deg);

return;

}

poly\_prod(frm, n >> 1, pos << 1);

poly\_prod(frm + (n >> 1), n - (n >> 1), pos << 1 | 1);

poly\_mul(mret[pos], mret[pos << 1], mret[pos << 1 | 1]);

}

void poly\_cprod(poly frm[], int n, int cst[], int pos = 1)

{

if(n == 1) {

poly\_const(msum[pos], cst[0]);

return;

}

poly\_cprod(frm, n >> 1, cst, pos << 1);

poly\_cprod(frm + (n >> 1), n - (n >> 1), cst + (n >> 1), pos << 1 | 1);

poly tmp0, tmp1;

poly\_mul(tmp0, mret[pos << 1], msum[pos << 1 | 1]);

poly\_mul(tmp1, msum[pos << 1], mret[pos << 1 | 1]);

poly\_add(msum[pos], tmp0, tmp1);

}

#ifdef FIX\_DEG

poly mds[1 << MAXL];

void \_eval\_proc(int ret[], int n, int pos = 1)

{

if(n == 1) {

ret[0] = mds[pos][0];

return;

}

poly\_mod(mds[pos << 1], mds[pos], mret[pos << 1]);

\_eval\_proc(ret, n >> 1, pos << 1);

poly\_mod(mds[pos << 1 | 1], mds[pos], mret[pos << 1 | 1]);

\_eval\_proc(ret + (n >> 1), n - (n >> 1), pos << 1 | 1);

}

void poly\_eval(int ret[], poly frm, int n, int pts[])

{

rep(i, n) poly\_line(mds[i], pts[i]);

poly\_prod(mds, n);

poly\_mod(mds[1], frm, mret[1]);

\_eval\_proc(ret, n);

}

void poly\_itpl(poly& ret, int n, int pts[], int val[])

{

rep(i, n) poly\_line(mds[i], pts[i]);

poly\_prod(mds, n);

poly cp;

int\* cur = new int[n];

poly\_diff(cp, mret[1]);

poly\_eval(cur, cp, n, pts);

gen\_inv(cur, n);

rep(i, n) cur[i] = LL(cur[i]) \* val[i] % MOD;

rep(i, n) poly\_line(mds[i], pts[i]);

poly\_cprod(mds, n, cur);

poly\_copy(ret, msum[1], msum[1].deg);

}

#endif

void poly\_power(poly& ret, poly x, int t, int d\_n)

{

int k = 0;

while(k < x.deg && x[k] == 0) k ++;

if(x[k] == 0) {

#ifdef FIX\_DEG

ret = poly(0);

#else

ret = poly(d\_n);

#endif

return;

}

int pt = x[k], ip = inv(pt);

poly rx(x.deg - k);

for(int i = k; i <= x.deg; i ++) rx[i - k] = LL(x[i]) \* ip % MOD;

poly ct;

poly\_ln(ct, rx, d\_n);

poly\_smul(rx, ct, t);

poly\_exp(ct, rx, d\_n);

ip = power(pt, t);

if(LL(k) \* t > d\_n) {

#ifdef FIX\_DEG

ret = poly(0);

#else

ret = poly(d\_n);

#endif

return;

}

ret = poly(d\_n);

for(int i = k \* t; i <= d\_n; i ++) ret[i] = LL(ct[i - k \* t]) \* ip % MOD;

#ifdef FIX\_DEG

while(ret.deg > 0 && ret[ret.deg] == 0) ret.deg --;

#endif

}

void poly\_shr(poly& ret, poly u, int d)

{

poly v, p, q;

p = poly(u.deg);

q = poly(u.deg);

for(int i = q.deg; i >= 0; i --) q[i] = i == q.deg ? 1 : LL(q[i + 1]) \* (MOD - d) % MOD;

for(int i = 0; i <= p.deg; i ++) p[i] = i == 0 ? 1 : LL(p[i - 1]) \* inv(i) % MOD;

for(int i = 0; i <= q.deg; i ++) q[i] = LL(q[i]) \* p[q.deg - i] % MOD;

for(int i = 0; i <= p.deg; i ++) p[i] = i == 0 ? 1 : LL(p[i - 1]) \* i % MOD;

for(int i = 0; i <= p.deg; i ++) p[i] = LL(p[i]) \* u[i] % MOD;

poly\_mul(v, p, q);

for(int i = 0; i <= p.deg; i ++) p[i] = i == 0 ? 1 : LL(p[i - 1]) \* inv(i) % MOD;

ret = poly(u.deg);

for(int i = 0; i <= ret.deg; i ++) ret[i] = LL(v[i + u.deg]) \* p[i] % MOD;

}

void poly\_dfac(poly& ret, int n)

{

if(n == 1) {

poly\_line(ret, 0);

return;

}

poly u, v;

poly\_dfac(u, n >> 1);

poly\_shr(v, u, n >> 1);

if(n & 1) {

poly w;

poly\_lmul(w, u, n - 1);

poly\_mul(ret, v, w);

} else poly\_mul(ret, u, v);

}

// 分治NTT的正确姿势（来自hdu6426）

//A(x)=x(A(x)^3/6+A(x)A(x^2)/2+A(x^3)/3)

inline void calc(int l,int r)

{

if (l+1==r) return;

int mid=(l+r)>>1,len=r-l;

calc(l,mid);

static int tmp[MAXN+48],a[MAXN+48],C1[MAXN+48],C2[MAXN+48];

poly::init(len<<1);

memset(a,0,poly::len\*sizeof(int));for (register int i=l;i<mid;i++) a[i-l]=A[i];

memset(C1,0,poly::len\*sizeof(int));for (register int i=0;i<len;i++) C1[i]=A[i];

memset(C2,0,poly::len\*sizeof(int));for (register int i=0;i\*2<len;i++) C2[i\*2]=A[i];

//A(x^3)

for (register int i=l;i\*3+1<r;i++) if (i\*3+1>=mid) A[i\*3+1]=add(A[i\*3+1]+(1ll\*A[i]\*I3)%MOD);

poly::ntt(a,1);poly::ntt(C1,1);poly::ntt(C2,1);

//A(x)^3

for (register int i=0;i<poly::len;i++) tmp[i]=(1ll\*a[i]\*C1[i]%MOD\*C1[i]%MOD);

poly::ntt(tmp,-1);

for (register int i=mid;i<r;i++) A[i]=add(A[i]+1ll\*tmp[i-l-1]\*(l?I2:I6)%MOD);

//A(x)A(x^2)

for (register int i=0;i<poly::len;i++) tmp[i]=(1ll\*a[i]\*C2[i])%MOD;

poly::ntt(tmp,-1);

for (register int i=mid;i<r;i++) A[i]=add(A[i]+1ll\*tmp[i-l-1]\*I2%MOD);

calc(mid,r);

}

//FWT

//fl=1表示FWT,fl=-1表示uFWT

void FWT(LL c[],int len,int fl)

{

int clen,j,k;

for (clen=2;clen<=len;clen<<=1)

for (j=0;j<len;j+=clen)

for (k=j;k<j+(clen>>1);k++)

{

LL tmp1=c[k],tmp2=c[k+(clen>>1)];

//对于xor

if (fl==1) c[k]=tmp1+tmp2,c[k+(clen>>1)]=tmp1-tmp2; else c[k]=(tmp1+tmp2)>>1,c[k+(clen>>1)]=(tmp1-tmp2)>>1;

//对于and

//if (fl==1) c[k]=tmp1+tmp2; else c[k]=tmp1-tmp2;

//对于or

//if (fl==1) c[k+(clen>>1)]=tmp1+tmp2; else c[k+(clen>>1)]=tmp2-tmp1;

}

}

void calc\_FWT()

{

FWT(a,1<<n,1);

FWT(b,1<<n,1);

for (int i=0;i<=(1<<n)-1;i++) a[i]\*=b[i];

FWT(a,1<<n,-1);

}