整除分块

void solve() {

ll n, m;

cin >> n >> m;

ll res = n \* 1ll \* m;

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

ll d = m / i, l = i, r;

if (d == 0) { r = n;} else { r = min(m / d, n); }

res -= d \* (l + r) \* (r - l + 1) / 2;

i = r;

}

cout << res << endl;

}

线性筛求积性函数

int n, m; int p[N], pr[N], pe[N], cnt;

int a[N], b[N], c[N], u[N]; // 因子个数、因子和、欧拉函数、莫比乌斯函数

void get\_primes(int n) {

p[1] = 1;

for (int i = 2; i <= n; i++) {

if (!p[i]) {

p[i] = i;

pe[i] = i;

pr[cnt++] = i;

}

for (int j = 0; i \* pr[j] <= n; j++) {

p[i \* pr[j]] = pr[j];

if (p[i] == pr[j]) {

pe[i \* pr[j]] = pe[i] \* pr[j];

break;

} else {

pe[i \* pr[j]] = pr[j];

}

}

}

}

int main() {

n = N - 1;

get\_primes(N - 1);

// 因子个数

a[1] = 1;

for (int i = 2; i <= n; i++) {

if (i == pe[i]) {

a[i] = a[i / p[i]] + 1;

} else {

a[i] = a[i / pe[i]] \* a[pe[i]];

}

}

// 因子和

b[1] = 1;

for (int i = 2; i <= n; i++) {

if (i == pe[i]) {

b[i] = b[i / p[i]] + i;

} else {

b[i] = b[i / pe[i]] \* b[pe[i]];

}

}

// 欧拉函数

c[1] = 1;

for (int i = 2; i <= n; i++) {

if (i == pe[i]) {

c[i] = i / p[i] \* (p[i] - 1);

} else {

c[i] = c[i / pe[i]] \* c[pe[i]];

}

}

// 莫比乌斯

u[1] = 1;

for (int i = 2; i <= n; i++) {

if (i == pe[i]) {

if (i == p[i]) u[i] = -1;

else u[i] = 0;

} else {

u[i] = u[i / pe[i]] \* u[pe[i]];

}

}

return 0;

}

线性基

int T, n, m, k;

ll num[N + 5];

bool insert(ll x) {

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

if (x & (1ll << i)) {

if (num[i] == 0) {

num[i] = x;

return true;

}

x ^= num[i];

}

}

return false;

}

ll query() {

ll ans = 0;

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

ans = max(ans, ans ^ num[i]);

}

return ans;

}

void solve() {

cin >> n;

for (int i = 1; i <= n; i++) { ll x; cin >> x; insert(x);}

cout << query() << endl;

}

扩展欧几里得

/\*

ax + by = c

d = exgcd(a, b, x, y)

x1 = x \* (c / d)

y1 = y \* (c / d)

-d / 2 <= t <= d / 2

x = x1 - (b / d) \* t

y = y1 + (a / d) \* t

最小正整数解 x = (x % (b / d) + (b / d)) % (b / d)

\*/

int exgcd(int a, int b, int &x, int &y) {

if (b == 0) {

x = 1;

y = 0;

return a;

}

int d = exgcd(b, a % b, y, x);

y -= a / b \* x;

return d;

}

int main() {

int a, b, c; cin >> a >> b >> c;

int x, y; int d = exgcd(a, b, x, y);

if (c % d != 0) {

cout << "-1" << endl;

} else {

int t = c / d;

cout << (-x \* t) << " " << (-y \* t) << endl;

}

return 0;

}

矩阵乘法快速幂

int n, m, k, x;

ll f[N], g[N][N];

void mul(ll a[][N], ll b[][N]) {

ll t[N][N] = {0};

for (int i = 0; i < x; i++)

for (int k = 0; k < x; k++)

if (a[i][k])

for (int j = 0; j < x; j++)

if (b[k][j])

t[i][j] = (t[i][j] + a[i][k] \* b[k][j]) % MOD;

memcpy(a, t, sizeof t);

}

void mul(ll a[], ll b[][N]) {

ll t[M] = {0};

for (int i = 0; i < x; i++)

for (int j = 0; j < x; j++)

t[i] = (t[i] + a[j] \* b[j][i]) % MOD;

memcpy(a, t, sizeof t);

}

int main() {

while (k) {

if (k & 1) mul(f, g);

mul(g, g);

k >>= 1;

}

return 0;

}

高斯消元

int T, n, m, k; double a[N][N];

void gauss() {

int l = 1;

for (int c = 1; c <= n; c++) {

int t = l;

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

if (fabs(a[i][c]) > fabs(a[t][c]))

t = i;

if (fabs(a[t][c]) < eps) continue;

for (int i = c; i <= n + 1; i++) swap(a[l][i], a[t][i]);

for (int i = n + 1; i >= c; i--) a[l][i] /= a[l][c];

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

if (fabs(a[i][c]) > eps) {

for (int j = n + 1; j >= c; j--) {

a[i][j] -= a[l][j] \* a[i][c];

}

}

}

++l;

}

if (l <= n) {

cout << "No Solution" << endl;

} else {

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

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

a[i][n + 1] -= a[i][j] \* a[j][n + 1];

}

}

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

printf("%.2f\n", a[i][n + 1]);

}

}

}

void solve() {

cin >> n;

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

for (int j = 1; j <= n + 1; j++) {

cin >> a[i][j];

}

}

gauss();

}

乘法逆元

// MOD 是质数

// inv[0] = inv[1] = 1

// inv[i] = (MOD - MOD / i) \* inv[MOD % i] % MOD

// inv[i] = pow(i, MOD - 2)

杜教筛

int pr[N], phi[N], cnt; bool st[N]; ll u[N], phi[N];

void init(int n) {

u[1] = phi[1] = 1;

for (int i = 2; i <= n; i++) {

if (!st[i]) {

u[i] = -1;

phi[i] = i - 1;

pr[cnt++] = i;

}

for (int j = 0; i \* pr[j] <= n; j++) {

st[i \* pr[j]] = true;

if (i % pr[j] == 0) {

u[i \* pr[j]] = 0;

phi[i \* pr[j]] = phi[i] \* pr[j];

break;

} else {

u[i \* pr[j]] = -u[i];

phi[i \* pr[j]] = phi[i] \* (pr[j] - 1);

}

}

}

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

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

phi[i] += phi[i - 1];

}

}

ll GetSumu(int n) {

if(n <= N) return sumu[n]; // sumu是提前筛好的前缀和

if(Smu[n]) return Smu[n]; // 记忆化

ll ret = 1ll; // 单位元的前缀和就是 1 (f \* 1 == ?)

for(ll l = 2, r; l <= n; l = r + 1) {

r = n / (n / l); ret -= (r - l + 1) \* GetSumu(n / l);

// (r - l + 1) 就是 I 在 [l, r] 的和

} return Smu[n] = ret; // 记忆化

}

ll GetSump(int n) {

if(n <= N) {

return sump[n]; // sumu是提前筛好的前缀和

}

if(Sphi[n]) return Sphi[n]; // 记忆化

ll ret = 1ll \* n \* (n + 1ll) / 2ll;

for(ll l = 2, r; l <= n; l = r + 1) {

r = n / (n / l); ret -= (r - l + 1) \* GetSump(n / l);

// (r - l + 1) 就是 I 在 [l, r] 的和

} return Sphi[n] = ret; // 记忆化

}

FFT

const int FFT\_MAXN = 262144 \* 8; const db pi = acosl(-1);

struct cp{

db a,b;

cp operator+(const cp&y)const{return (cp){a+y.a,b+y.b};}

cp operator-(const cp&y)const{return (cp){a-y.a,b-y.b};}

cp operator\*(const cp&y)const{return (cp){a\*y.a-b\*y.b,a\*y.b+b\*y.a};}

cp operator! ()const{return (cp){a,-b};};

}nw[FFT\_MAXN+1];int bitrev[FFT\_MAXN];

void dft(cp\*a,int n,int flag=1){

int d=0;while((1<<d)\*n!=FFT\_MAXN)d++;

for(int i=0;i<n;i++)if(i<(bitrev[i]>>d))swap(a[i],a[bitrev[i]>>d]);

for (int l=2;l<=n;l<<=1){

int del=FFT\_MAXN/l\*flag;

for (int i=0;i<n;i+=l){

cp \*le=a+i,\*ri=a+i+(l>>1),\*w=flag==1?nw:nw+FFT\_MAXN;

for(int k=0;k<l>>1;k++){

cp ne=\*ri\*\*w;

\*ri=\*le-ne,\*le=\*le+ne;

le++,ri++,w+=del;

}

}

}

if(flag!=1)for(int i=0;i<n;i++)a[i].a/=n,a[i].b/=n;

}

void fft\_init(){

int L=0;while((1<<L)!=FFT\_MAXN)L++;

bitrev[0]=0; for(int i=1;i<FFT\_MAXN;i++)bitrev[i]=bitrev[i>>1]>>1| ( ( i&1)<<(L-1) );

nw[0]=nw [FFT\_MAXN]=(cp){1,0};

for(int i=0;i<FFT\_MAXN+1;i++)nw[i]=(cp){cosl(2\*pi/FFT\_MAXN\*i),sinl(2\*pi/FFT\_MAXN\*i)};

}

void cpnvo(db \*a,int n,db\*b,int m,db\*c){

static cp f[FFT\_MAXN>>1],g[FFT\_MAXN>>1],t[FFT\_MAXN>>1];

int N=2;while(N<=n+m)N<<=1;

for(int i=0;i<N;i++)

if(i&1){

f[i>>1].b=(i<=n)?a[i]:0.0;

g[i>>1].b=(i<=m)?b[i]:0.0;

}else{

f[i>>1].a=(i<=n)?a[i]:0.0;

g[i>>1].a=(i<=m)?b[i]:0.0;

}

dft(f,N>>1);dft(g,N>>1);

int del=FFT\_MAXN/(N>>1);

cp qua=(cp){0,0.25}, one=(cp){1,0}, four=(cp){4,0}, \*w=nw;

for(int i=0;i<N>>1;i++){

int j=i?(N>>1)-i:0;

t[i]=(four\*!(f[j]\*g[j])-(!f[j]-f[i])\*(!g[j]-g[i])\*(one+\*w))\*qua;

w+=del;

}

dft(t,N>>1,-1);

for(int i=0;i<n+m+1;i++)c[i]=(i&1)?t[i>>1].a:t[i>>1].b;

}

NTT

int n, m;

struct Complex {

double x, y;

Complex operator+ (const Complex& t) const {

return {x + t.x, y + t.y};

}

Complex operator- (const Complex& t) const {

return {x - t.x, y - t.y};

}

Complex operator\* (const Complex& t) const {

return {x \* t.x - y \* t.y, x \* t.y + y \* t.x};

}

}a[N], b[N];

int rev[N], bit, tot;

void fft(Complex a[], int inv) {

for (int i = 0; i < tot; i ++ )

if (i < rev[i])

swap(a[i], a[rev[i]]);

for (int mid = 1; mid < tot; mid <<= 1) {

auto w1 = Complex({cos(PI / mid), inv \* sin(PI / mid)});

for (int i = 0; i < tot; i += mid \* 2) {

auto wk = Complex({1, 0});

for (int j = 0; j < mid; j ++, wk = wk \* w1) {

auto x = a[i + j], y = wk \* a[i + j + mid];

a[i + j] = x + y, a[i + j + mid] = x - y;

}

}

}

}

int main()

{

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

for (int i = 0; i <= n; i ++ ) scanf("%lf", &a[i].x);

for (int i = 0; i <= m; i ++ ) scanf("%lf", &b[i].x);

while ((1 << bit) < n + m + 1) bit ++;

tot = 1 << bit;

for (int i = 0; i < tot; i ++ )

rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (bit - 1));

fft(a, 1), fft(b, 1);

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

fft(a, -1);

for (int i = 0; i <= n + m; i ++ )

printf("%d ", (int)(a[i].x / tot + 0.5));

return 0;

}