namespace LCT

{

#define ls(x) (c[x][0])

#define rs(x) (c[x][1])

#define s(x, k) (c[x][k])

int val[N], f[N], siz[N], si[N];

long long siz2[N], si2[N]; // si 轻儿子

int c[N][2];

inline long long sqr(int x)

{

return 1ll \* x \* x;

}

inline bool ws(int x, int p)

{

return ls(p) ^ x;

}

inline bool nroot(int x)

{

return ls(f[x]) == x || rs(f[x]) == x;

}

inline void pushup(int x)

{

siz[x] = siz[ls(x)] + siz[rs(x)] + si[x] + 1;

siz2[x] = sqr(siz[ls(x)]) + sqr(siz[rs(x)]) + si2[x];

}

inline void rotate(int x)

{

int p = f[x], g = f[p];

int t = ws(x, p), w = s(x, !t);

if (nroot(p)) s(g, ws(p, g)) = x;

s(x, !t) = p;

s(p, t) = w;

if (w) f[w] = p;

f[p] = x;

f[x] = g;

pushup(p);

}

inline void Splay(int x)

{

while (nroot(x))

{

int p = f[x], g = f[p];

if (nroot(p)) rotate(ws(x, p) ^ ws(p, g) ? x : p);

rotate(x);

}

pushup(x);

}

inline void access(int x)

{

for (int y = 0; x; x = f[y = x])

Splay(x), si[x] += siz[rs(x)] - siz[y], si2[x] += sqr(siz[rs(x)]) - sqr(siz[y]), rs(x) = y,

pushup(x);

}

inline int findroot(int x)

{

access(x), Splay(x);

while (ls(x)) x = ls(x);

return Splay(x), x;

}

inline void link(int u, int v)

{

access(v), Splay(v);

Splay(u);

si[v] += siz[u], si2[v] += sqr(siz[u]);

pushup(f[u] = v);

}

inline void cut(int u, int v)

{

access(v), Splay(v);

Splay(u);

si[v] -= siz[u], si2[v] -= sqr(siz[u]), f[u] = 0;

pushup(v);

}

} // namespace LCT

using namespace LCT;

namespace Tarjan

{

int dfn[N], low[N], dfncnt, s[N], in\_stack[N], tp;

int scc[N], sc; // 结点 i 所在 SCC 的编号

int sz[N]; // 强连通 i 的大小

void tarjan(int u)

{

low[u] = dfn[u] = ++dfncnt, s[++tp] = u, in\_stack[u] = 1;

for (int i = h[u]; i; i = e[i].nex)

{

const int &v = e[i].t;

if (!dfn[v])

{

tarjan(v);

low[u] = min(low[u], low[v]);

}

else if (in\_stack[v])

{

low[u] = min(low[u], dfn[v]);

}

}

if (dfn[u] == low[u])

{

++sc;

while (s[tp] != u)

{

scc[s[tp]] = sc;

sz[sc]++;

in\_stack[s[tp]] = 0;

--tp;

}

scc[s[tp]] = sc;

sz[sc]++;

in\_stack[s[tp]] = 0;

--tp;

}

}

} // namespace Tarjan

namespace Euler\_Circuit

{

struct edge

{

int to;

bool exists;

int revref;

bool operator<(const edge &b) const

{

return to < b.to;

}

};

vector<edge> beg[505];

int cnt[505];

const int dn = 500;

stack<int> ans;

void Hierholzer(int x)

{ // 关键函数

for (int &i = cnt[x]; i < (int)beg[x].size();)

{

if (beg[x][i].exists)

{

edge e = beg[x][i];

beg[x][i].exists = 0;

beg[e.to][e.revref].exists = 0;

++i;

Hierholzer(e.to);

}

else

{

++i;

}

}

ans.push(x);

}

int deg[505];

int reftop[505];

int main()

{

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

{

beg[i].reserve(1050); // vector 用 reserve 避免动态分配空间，加快速度

}

int m;

scanf("%d", &m);

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

{

int a, b;

scanf("%d%d", &a, &b);

beg[a].push\_back((edge){b, 1, 0});

beg[b].push\_back((edge){a, 1, 0});

++deg[a];

++deg[b];

}

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

{

if (!beg[i].empty())

{

sort(beg[i].begin(), beg[i].end()); // 为了要按字典序贪心，必须排序

}

}

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

{

for (int j = 0; j < (int)beg[i].size(); ++j)

{

beg[i][j].revref = reftop[beg[i][j].to]++;

}

}

int bv = 0;

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

{

if (!deg[bv] && deg[i])

{

bv = i;

}

else if (!(deg[bv] & 1) && (deg[i] & 1))

{

bv = i;

}

}

Hierholzer(bv);

while (!ans.empty())

{

printf("%d\n", ans.top());

ans.pop();

}

}

} // namespace Euler\_Circuit

namespace Netflow

{

// Dinic

struct node

{

int to, net;

long long val;

} e[520010];

inline void add(int u, int v, long long w)

{

e[++tot].to = v;

e[tot].val = w;

e[tot].net = head[u];

head[u] = tot;

e[++tot].to = u;

e[tot].val = 0;

e[tot].net = head[v];

head[v] = tot;

}

inline int bfs()

{ // 在惨量网络中构造分层图

for (register int i = 1; i <= n; i++) dis[i] = inf;

queue<int> q;

q.push(s);

dis[s] = 0;

now[s] = head[s];

while (!q.empty())

{

int x = q.front();

q.pop();

for (register int i = head[x]; i; i = e[i].net)

{

int v = e[i].to;

if (e[i].val > 0 && dis[v] == inf)

{

q.push(v);

now[v] = head[v];

dis[v] = dis[x] + 1;

if (v == t) return 1;

}

}

}

return 0;

}

inline int dfs(int x, long long sum)

{ // sum是整条增广路对最大流的贡献

if (x == t) return sum;

long long k, res = 0; // k是当前最小的剩余容量

for (register int i = now[x]; i && sum; i = e[i].net)

{

now[x] = i; // 当前弧优化

int v = e[i].to;

if (e[i].val > 0 && (dis[v] == dis[x] + 1))

{

k = dfs(v, min(sum, e[i].val));

if (k == 0) dis[v] = inf; // 剪枝，去掉增广完毕的点

e[i].val -= k;

e[i ^ 1].val += k;

res += k; // res表示经过该点的所有流量和（相当于流出的总量）

sum -= k; // sum表示经过该点的剩余流量

}

}

return res;

}

void add(int u, int v, int c, int w)

{

E[++tot].next = head[u];

E[tot].to = v;

E[tot].c = c;

E[tot].w = w;

head[u] = tot;

E[++tot].next = head[v];

E[tot].to = u;

E[tot].c = 0;

E[tot].w = -w;

head[v] = tot;

}

bool SPFA()

{

queue<int> q;

memset(dis, 0x3f, sizeof(dis));

memset(flow, 0x3f, sizeof(flow));

q.push(S);

dis[S] = 0;

vis[S] = 1;

while (!q.empty())

{

int u = q.front();

q.pop();

for (int i = head[u]; ~i; i = E[i].next)

{

int v = E[i].to;

int c = E[i].c;

int w = E[i].w;

if (c && dis[v] > dis[u] + w)

{

dis[v] = dis[u] + w;

pre[v] = u;

cur[v] = i;

flow[v] = min(c, flow[u]);

if (!vis[v])

{

q.push(v);

vis[v] = 1;

}

}

}

vis[u] = 0;

}

return flow[S] != flow[T];

}

pair<int, int> min\_cost\_max\_flow()

{

int max\_flow = 0;

int min\_cost = 0;

while (SPFA())

{

max\_flow += flow[T];

min\_cost += dis[T] \* flow[T];

for (int u = T; u != S; u = pre[u])

{

int v = cur[u];

E[v].c -= flow[T];

E[v ^ 1].c += flow[T];

}

}

return make\_pair(max\_flow, min\_cost);

}

} // namespace Netflow

/\*

@Date : 2020-09-30 17:28:10

@Author : Adscn (adscn@qq.com)

@Link : https://www.cnblogs.com/LLCSBlog

\*/

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

#define IL inline

#define RG register

#define gi geti<int>()

#define gl geti<ll>()

#define gc getchar()

#define File(a) \

freopen(a ".in", "r", stdin); \

freopen(a ".out", "w", stdout)

template<typename T>

IL bool chkmax(T &x, const T &y)

{

return x < y ? x = y, 1 : 0;

}

template<typename T>

IL bool chkmin(T &x, const T &y)

{

return x > y ? x = y, 1 : 0;

}

template<typename T>

IL T geti()

{

RG T xi = 0;

RG char ch = gc;

bool f = 0;

while (!isdigit(ch)) ch == '-' ? f = 1 : f, ch = gc;

while (isdigit(ch)) xi = xi \* 10 + ch - 48, ch = gc;

return f ? -xi : xi;

}

template<typename T>

IL void pi(T k, char ch = 0)

{

if (k < 0) k = -k, putchar('-');

if (k >= 10) pi(k / 10);

putchar(k % 10 + '0');

if (ch) putchar(ch);

}

// #define FFT\_

// #define FAST

// #define SECURE

#ifdef FFT\_

# ifdef FAST

# define FAST\_FAST\_TLE\_

# endif

# ifdef SECURE

# define HIGH\_PRECISION

# endif

#endif

#ifdef FAST

# include <bits/extc++.h>

#endif

namespace Math

{

const int P = 998244353, G = 3, Gi = 332748118;

typedef vector<int> Vec;

typedef unsigned long long ull;

inline ll fpow(ll a, int b, int MOD = P, ll c = 1)

{

for (; b; b >>= 1, a = a \* a % MOD)

if (b & 1) c = c \* a % MOD;

return c;

}

inline int inv(int x, int MOD = P)

{

return fpow(x, MOD - 2);

}

inline int extend(int x)

{

int n = 1;

while (n < x) n <<= 1;

return n;

}

inline int add(int x)

{

return x > P ? x - P : x;

}

inline int sub(int x)

{

return x < 0 ? x + P : x;

}

inline void add(int &x, int y)

{

(x += y) >= P && (x -= P);

}

inline void sub(int &x, int y)

{

(x -= y) < 0 && (x += P);

}

inline int BSGS(int a, int b, int p)

{

#ifdef FAST

\_\_gnu\_pbds::gp\_hash\_table<int, int> mp;

#else

unordered\_map<int, int> mp;

#endif

int m = ceil(sqrt(p)); // ceil!!

for (int i = 0; i <= m; b = 1ll \* a \* b % p, ++i) mp[b] = i;

a = fpow(a, m);

for (int i = 0, j = 1; i < m; j = 1ll \* j \* a % p)

if (mp.find(j) != mp.end() && i \* m >= mp[j]) return i \* m - mp[j];

return -1;

}

inline int root(int p)

{

Vec fac;

int phi = p - 1, x = phi;

for (int i = 2; i \* i <= x; ++i)

if (x % i == 0)

{

fac.push\_back(i);

while (x % i == 0) x /= i;

}

if (x > 1) fac.push\_back(x);

for (int i = 2; i <= phi; ++i)

{

bool flag = 1;

for (auto j : fac)

if (fpow(i, phi / j, p) == 1)

{

flag = 0;

break;

}

if (flag) return i;

}

return -1;

}

int degree(int a, int k, int p)

{

int g = root(p);

int x = BSGS(g, a, p);

assert(x >= 0 && x % k == 0);

int r = fpow(g, x / k, p);

return min(r, p - r);

}

} // namespace Math

using namespace Math;

namespace Poly

{

#define lg2(x) (31 - \_\_builtin\_clz(x))

namespace FFT

{

#ifdef HIGH\_PRECISION

typedef \_\_float128 db;

#else

typedef double db;

#endif

const db PI = acos(-1);

struct complex

{

db x, y;

complex() {}

explicit complex(db \_x, db \_y)

: x(\_x)

, y(\_y)

{

}

};

typedef vector<complex> Vex;

inline complex operator+(const complex &a, const complex &b)

{

return complex(a.x + b.x, a.y + b.y);

}

inline complex operator-(const complex &a, const complex &b)

{

return complex(a.x - b.x, a.y - b.y);

}

inline complex operator\*(const complex &a, const complex &b)

{

return complex(a.x \* b.x - a.y \* b.y, a.x \* b.y + a.y \* b.x);

}

inline complex &operator+=(complex &a, const complex &b)

{

return a = a + b;

}

inline complex &operator-=(complex &a, const complex &b)

{

return a = a - b;

}

inline complex &operator\*=(complex &a, const complex &b)

{

return a = a \* b;

}

inline void FFT(Vex &a, int opt)

{

int n = a.size(), L = lg2(n);

Vec R(n);

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

{

R[i] = R[i >> 1] >> 1 | (i & 1) << (L - 1);

if (i < R[i]) swap(a[i], a[R[i]]);

}

for (int len = 1; len < n; len <<= 1)

{

complex O = complex(cos(PI / len), opt \* sin(PI / len));

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

{

complex o = complex(1, 0);

for (int j = 0; j < len; ++j, o = o \* O)

{

complex Nx = a[i + j], Ny = o \* a[i + j + len];

a[i + j] = Nx + Ny, a[i + j + len] = Nx - Ny;

}

}

}

}

inline void DFT(Vex &a)

{

FFT(a, 1);

}

inline void IDFT(Vex &a)

{

FFT(a, -1);

int n = a.size();

for (auto &i : a)

#ifdef FAST\_FAST\_TLE\_

i.y /= 2 \* n;

#else

i.x /= n;

#endif

}

Vec mul(Vec a, Vec b)

{

int n = a.size() + b.size() - 1, N = extend(n);

a.resize(N), b.resize(N);

#ifdef FAST\_FAST\_TLE\_

Vex A(N);

for (int i = 0; i < N; ++i) A[i] = complex(a[i], b[i]);

DFT(A);

for (auto &i : A) i \*= i;

IDFT(A);

for (int i = 0; i < N; ++i) a[i] = (int)(A[i].y + 0.5);

#else

Vex A(N), B(N);

for (int i = 0; i < N; ++i) A[i].x = a[i], B[i].x = b[i];

DFT(A), DFT(B);

for (int i = 0; i < N; ++i) A[i] \*= B[i];

IDFT(A);

for (int i = 0; i < N; ++i) a[i] = (int)(A[i].x + 0.5);

#endif

return a.resize(n), a;

}

} // namespace FFT

namespace NTT

{

inline void NTT(Vec &a, int opt)

{

int n = a.size(), L = lg2(n);

Vec R(n);

vector<ull> t(n);

copy(a.begin(), a.end(), t.begin());

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

{

R[i] = R[i >> 1] >> 1 | (i & 1) << (L - 1);

if (i < R[i]) swap(t[i], t[R[i]]);

}

for (int len = 1; len < n; len <<= 1)

{

ll O = fpow(opt == 1 ? G : Gi, (P - 1) / (len << 1));

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

{

ll o = 1;

for (int j = 0; j < len; ++j, o = o \* O % P)

{

int tmp = o \* t[i + j + len] % P;

t[i + j + len] = t[i + j] - tmp + P;

t[i + j] += tmp;

}

}

}

for (int i = 0; i < n; ++i) a[i] = t[i] % P;

}

inline void DFT(Vec &a)

{

NTT(a, 1);

}

inline void IDFT(Vec &a)

{

NTT(a, -1);

int Inv = inv(a.size());

for (auto &i : a) i = 1ll \* i \* Inv % P;

}

Vec mul(Vec a, Vec b)

{

int n = a.size() + b.size() - 1, N = extend(n);

a.resize(N), b.resize(N);

DFT(a), DFT(b);

for (int i = 0; i < N; ++i) a[i] = 1ll \* a[i] \* b[i] % P;

IDFT(a);

return a.resize(n), a;

}

} // namespace NTT

#ifdef FFT\_

using FFT::mul;

using FFT::DFT;

using FFT::IDFT;

#else

using NTT::mul;

using NTT::DFT;

using NTT::IDFT;

#endif

Vec operator\*(const Vec &a, const Vec &b)

{

return mul(a, b);

}

Vec &operator\*=(Vec &a, const Vec &b)

{

return a = a \* b;

}

#ifndef FFT\_

Vec operator~(Vec);

Vec fix(Vec, int);

Vec operator-(Vec A, Vec B)

{

int n = max(A.size(), B.size());

A.resize(n), B.resize(n);

for (int i = 0; i < n; ++i) A[i] = sub(A[i] - B[i]);

return A;

}

Vec operator-(Vec A)

{

for (auto i : A) i = sub(-i);

return A;

}

Vec operator-(int v, Vec A)

{

sub(A[0], v);

return A;

}

Vec operator-(Vec A, int v)

{

sub(A[0], v);

return A;

}

Vec operator+(int v, Vec A)

{

add(A[0], v);

return A;

}

Vec operator+(Vec A, int v)

{

add(A[0], v);

return A;

}

Vec operator+(Vec A, Vec B)

{

int n = max(A.size(), B.size());

A.resize(n), B.resize(n);

for (int i = 0; i < n; ++i) A[i] = add(A[i] + B[i]);

return A;

}

Vec operator/(Vec A, Vec B)

{

int n = A.size() - B.size() + 1;

if (n <= 0) return Vec(1, 0);

reverse(A.begin(), A.end());

reverse(B.begin(), B.end());

A.resize(n), B.resize(n);

A = fix(A \* ~B, n);

return reverse(A.begin(), A.end()), A;

}

Vec operator%(Vec A, Vec B)

{

int n = B.size() - 1;

return fix(A - A / B \* B, n);

}

Vec operator~(Vec A)

{

int n = A.size(), N = extend(n);

A.resize(N);

Vec I(N);

I[0] = inv(A[0]);

for (int l = 2; l <= N; l <<= 1)

{

Vec X(l), Y(l);

copy(A.begin(), A.begin() + l, X.begin());

copy(I.begin(), I.begin() + l, Y.begin());

int L = l << 1;

X.resize(L), Y.resize(L);

DFT(X), DFT(Y);

for (int i = 0; i < L; ++i) X[i] = 1ll \* Y[i] \* (P + 2 - 1ll \* X[i] \* Y[i] % P) % P;

IDFT(X), X.resize(l);

copy(X.begin(), X.begin() + l, I.begin());

}

return I.resize(n), I;

}

Vec fix(Vec A, int n)

{

return A.resize(n), A;

}

Vec der(Vec A, bool mod = 1)

{

int n = A.size();

if (n == 1) return Vec(1, 0);

Vec D(n - 1);

for (int i = 1; i < n; ++i) D[i - 1] = 1ll \* i \* A[i] % P;

if (mod) D.resize(n);

return D;

}

Vec inte(Vec A, bool mod = 1)

{

int n = A.size();

Vec I = Vec(n + 1, 0);

for (int i = 1; i <= n; ++i) I[i] = 1ll \* inv(i) \* A[i - 1] % P;

if (mod) I.resize(n);

return I;

}

inline Vec ln(Vec A)

{

assert(A[0] == 1);

int n = A.size();

return inte(fix(der(A) \* ~A, n));

}

inline Vec sqrt(Vec A)

{

int n = A.size(), N = extend(n);

A.resize(N);

Vec R(N);

R[0] = degree(A[0], 2, P);

int i2 = inv(2, P);

for (int l = 2; l <= N; l <<= 1)

{

Vec F(l), G(l);

copy(A.begin(), A.begin() + l, F.begin());

copy(R.begin(), R.begin() + l, G.begin());

Vec I = ~G;

int L = l << 1;

F.resize(L), G.resize(L), I.resize(L);

DFT(F), DFT(G), DFT(I);

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

G[i] = 1ll \* G[i] \* G[i] % P, F[i] = 1ll \* (F[i] + G[i]) \* i2 % P \* I[i] % P;

IDFT(F), F.resize(l);

copy(F.begin(), F.begin() + l, R.begin());

}

return R.resize(n), R;

}

inline Vec exp(Vec A)

{

assert(!A[0]);

int n = A.size(), N = extend(n);

A.resize(N);

Vec E(N, 0);

E[0] = 1;

for (int l = 2; l <= N; l <<= 1)

{

Vec P = (fix(A, l) - ln(fix(E, l)) + 1) \* fix(E, l);

copy(P.begin(), P.begin() + l, E.begin());

}

return E.resize(n), E;

}

Vec ls, rs, pos;

vector<Vec> h, g;

int id;

# define mid ((l + r) >> 1)

inline void dfs(int &rt, int l, int r)

{

rt = id++;

if (l == r) return pos[l] = rt, void();

dfs(ls[rt], l, mid), dfs(rs[rt], mid + 1, r);

}

# undef mid

inline void build(int n)

{

ls.resize(2 \* n), rs.resize(2 \* n);

h.resize(2 \* n), g.resize(2 \* n);

pos.resize(n), id = 0;

dfs(\*new int, 0, n - 1);

}

pair<Vec, Vec> calc(Vec H, Vec G1, Vec G2)

{

int n = H.size(), N = extend(n), m1 = G1.size(), m2 = G2.size();

reverse(G1.begin(), G1.end());

reverse(G2.begin(), G2.end());

H.resize(N), G1.resize(N), G2.resize(N);

DFT(H), DFT(G1), DFT(G2);

for (int i = 0; i < N; ++i) G1[i] = 1ll \* G1[i] \* H[i] % P;

for (int i = 0; i < N; ++i) G2[i] = 1ll \* G2[i] \* H[i] % P;

IDFT(G1), IDFT(G2);

return make\_pair(Vec(G2.begin() + m2 - 1, G2.begin() + n),

Vec(G1.begin() + m1 - 1, G1.begin() + n));

}

Vec \_eval(Vec A, Vec Q)

{

int n = A.size(), N = extend(2 \* n - 2);

build(n);

for (int i = 0; i < n; ++i) g[pos[i]] = (Vec){1, add(P - Q[i])};

for (int i = n \* 2 - 2; ~i; --i)

if (ls[i]) g[i] = g[ls[i]] \* g[rs[i]];

g[0] = ~g[0];

reverse(g[0].begin(), g[0].end());

A.resize(N), g[0].resize(N);

DFT(A), DFT(g[0]);

for (int i = 0; i < N; ++i) A[i] = 1ll \* A[i] \* g[0][i] % P;

IDFT(A);

h[0].resize(n);

copy(A.begin() + n, A.begin() + 2 \* n, h[0].data());

for (int i = 0; i < n \* 2 - 1; ++i)

if (ls[i]) tie(h[ls[i]], h[rs[i]]) = calc(h[i], g[ls[i]], g[rs[i]]);

Vec ans(n);

for (int i = 0; i < n; ++i) ans[i] = h[pos[i]][0];

return ans;

}

Vec eval(Vec A, Vec Q)

{

ls.clear(), rs.clear(), pos.clear(), h.clear(), g.clear();

int n = A.size(), m = Q.size();

A.resize(max(n, m)), Q.resize(max(n, m));

return fix(\_eval(A, Q), m);

}

#endif

} // namespace Poly

using namespace Poly;

int main(void)

{

int n = gi, m = gi;

Vec a(n + 1), b(m);

for (int i = 0; i <= n; ++i) a[i] = gi;

for (int i = 0; i < m; ++i) b[i] = gi;

Vec ans = eval(a, b);

for (auto i : ans) pi(i, '\n');

return 0;

}

const int MAXN = 4000000;

const int N = INT\_MAX;

const int SQRN = sqrt(N) + 7;

typedef long long ll;

int n, sqrn;

bool npr[MAXN + 7];

int pr[MAXN + 7], cnt;

ll mu[MAXN + 7];

unsigned long long phi[MAXN + 7];

ll dpp[SQRN << 1], dpm[SQRN << 1];

// map<int,ll>dpp,dpm;

void init()

{

npr[1] = 1;

mu[1] = phi[1] = 1;

for (int i = 2; i <= MAXN; ++i)

{

if (!npr[i]) pr[++cnt] = i, mu[i] = -1, phi[i] = i - 1;

for (int j = 1; j <= cnt && 1ll \* pr[j] \* i <= MAXN; ++j)

{

npr[i \* pr[j]] = 1;

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

{

mu[i \* pr[j]] = 0, phi[i \* pr[j]] = phi[i] \* pr[j];

break;

}

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

}

}

for (int i = 1; i <= MAXN; ++i) mu[i] += mu[i - 1], phi[i] += phi[i - 1];

}

inline ll getphi(int x)

{

if (x <= MAXN) return phi[x];

if (x <= SQRN && dpp[x])

return dpp[x];

else if (x > SQRN && dpp[N / x + SQRN])

return dpp[N / x + SQRN];

unsigned long long ans = 1ull \* x \* (x + 1ull) / 2ull;

for (int l = 2, r; r < INT\_MAX && l <= x; l = r + 1)

r = x / (x / l), ans -= 1ull \* (r - l + 1) \* getphi(x / l);

if (x <= SQRN)

dpp[x] = ans;

else

dpp[N / x + SQRN] = ans;

return ans;

}

inline ll getmu(int x)

{

if (x <= MAXN) return mu[x];

if (x <= SQRN && dpm[x])

return dpm[x];

else if (x > SQRN && dpm[N / x + SQRN])

return dpm[N / x + SQRN];

ll ans = 1;

for (int l = 2, r; r < INT\_MAX && l <= x; l = r + 1)

r = x / (x / l), ans -= 1ll \* (r - l + 1) \* getmu(x / l);

if (x <= SQRN)

dpm[x] = ans;

else

dpm[N / x + SQRN] = ans;

return ans;

}

double simpson(double (&f)(double), double l, double r)

{

double mid = (l + r) / 2;

return (r - l) \* (f(l) + 4 \* f(mid) + f(r)) / 6;

}

double asr(double (&f)(double), double l, double r, double ans)

{

double mid = (l + r) / 2;

double left = simpson(f, l, mid);

double right = simpson(f, mid, r);

if (fabs(left + right - ans) < eps) return left + right;

return asr(f, l, mid, left) + asr(f, mid, r, right);

}

template<typename \_Tp>

bool gauss(\_Tp \*\*equation\_set, int n)

{

const \_Tp eps = 1e-20;

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

{

int cur = i;

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

if (abs(equation\_set[j][i]) > abs(equation\_set[cur][i])) cur = i;

swap(equation\_set[i], equation\_set[cur]);

for (int j = n; j >= i; --j)

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

{

if (abs(equation\_set[i][i]) <= eps) return 0;

equation\_set[k][j] -= equation\_set[k][i] / equation\_set[i][i] \* equation\_set[i][j];

}

}

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

{

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

equation\_set[i][n] -= equation\_set[j][n] \* equation\_set[i][j];

if (abs(equation\_set[i][i]) <= eps) return 0;

equation\_set[i][n] /= equation\_set[i][i];

}

return 1;

}

namespace geo

{

struct Vector

{

double x, y;

};

using Point = Vector;

bool operator==(const Vector &a, const Vector &b)

{

return !(dcmp(a.x - b.x) && dcmp(a.y - b.y));

}

Vector operator+(const Vector &a, const Vector &b)

{

return {a.x + b.x, a.y + b.y};

}

Vector operator-(const Vector &a, const Vector &b)

{

return {a.x - b.x, a.y - b.y};

}

template<typename T>

Vector operator\*(const Vector &a, const T &b)

{

return {a.x \* b, a.y \* b};

}

template<typename T>

Vector operator/(const Vector &a, const T &b)

{

return {a.x / b, a.y / b};

}

double Dot(Vector a, Vector b)

{

return a.x \* b.x + a.y \* b.y;

}

double Length(Vector x)

{

return sqrt(x.x \* x.x + x.y \* x.y);

}

double Cross(Vector x, Vector y)

{

return x.x \* y.y - x.y \* y.x;

}

inline int dcmp(double val)

{

if (fabs(val) < eps)

return 0;

else

return val > 0 ? 1 : -1;

}

double DisTS(Point P, Point A, Point B)

{ // Distance\_to\_Segment

if (A == B) return Length(P - A);

Vector V1 = B - A, V2 = P - A, V3 = P - B;

if (dcmp(Dot(V1, V2)) < 0)

return Length(V2);

else if (dcmp(Dot(V1, V3)) > 0)

return Length(V3);

else

return fabs(Cross(V1, V2) / Length(V1));

}

inline int ConvexHull(Point A[], int n, Point Ans[])

{

std::sort(A + 1, A + n + 1);

int cnt = 0;

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

{

while (cnt > 1 && Cross(Ans[cnt] - Ans[cnt - 1], A[i] - Ans[cnt - 1]) <= 0) cnt--;

Ans[++cnt] = A[i];

}

int k = cnt;

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

{

while (cnt > k && Cross(Ans[cnt] - Ans[cnt - 1], A[i] - Ans[cnt - 1]) <= 0) cnt--;

Ans[++cnt] = A[i];

}

return cnt - 1;

}

inline double Rotating\_Calipers(Point A[], int n, Point Poly[])

{

n = ConvexHull(A, n, Poly);

Poly[n + 1] = Poly[1];

double ans = 0;

for (int u = 1, v = 2; u <= n; u++)

{

while (true)

{

int val = dcmp(Cross(Poly[u + 1] - Poly[u], Poly[v + 1] - Poly[v]));

if (val <= 0)

{

ans = std::max(ans, Length(Poly[u] - Poly[v]));

if (!val) ans = std::max(ans, Length(Poly[u] - Poly[v + 1]));

break;

}

v = v % n + 1;

}

}

return ans;

}

inline double area(Point a, Point b, Point c)

{

return fabs(Cross(b - a, c - a));

}

struct Line

{

Point P;

Vector V;

double ang;

Line(Point a = Point(0, 0), Vector b = Vector(0, 0))

: P(a)

, V(b)

{

ang = atan2(V.y, V.x);

} // 极角

bool operator<(const Line &rhs) const

{

return ang < rhs.ang;

}

};

bool operator<(const Line &a, const Line &b)

{

return a.ang < b.ang;

}

inline Point LI(Line a, Line b)

{

Vector V = a.P - b.P;

return a.P + a.V \* (Cross(b.V, V) / Cross(a.V, b.V));

}

inline bool OnLeft(Line a, Point b)

{

return Cross(b - a.P, a.V) < 0;

}

int HpI(Line L[], int n, Point Poly[])

{

std::sort(L + 1, L + n + 1);

int f, l;

static Point p[N];

static Line q[N];

q[f = l = 1] = L[1];

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

{

while (f < l && !OnLeft(L[i], p[l - 1]))

l--; // 如果原来的交点在新加入直线的右边，就没用了。

while (f < l && !OnLeft(L[i], p[f])) f++;

q[++l] = L[i];

if (!dcmp(Cross(q[l].V, q[l - 1].V)))

{ // 与上一条直线极角相同

l--;

if (OnLeft(q[l], L[i].P)) q[l] = L[i];

}

if (f < l) p[l - 1] = LI(q[l], q[l - 1]); // 新的交点

}

while (f < l && !OnLeft(q[f], p[l - 1])) l--;

if (l - f <= 1) return 0;

p[l] = LI(q[l], q[f]);

int cnt = 0;

for (int i = f; i <= l; i++) Poly[++cnt] = p[i];

return cnt;

}

inline double sqr(double x)

{

return x \* x;

}

inline double dist(Point x, Point y)

{

return sqrt(sqr(y.x - x.x) + sqr(y.y - x.y));

}

Point Center(Point a, Point b, Point c)

{

double a1 = b.x - a.x, b1 = b.y - a.y, c1 = (sqr(a1) + sqr(b1)) / 2;

double a2 = c.x - a.x, b2 = c.y - a.y, c2 = (sqr(a2) + sqr(b2)) / 2;

double d = a1 \* b2 - a2 \* b1;

return Point{a.x + (c1 \* b2 - c2 \* b1) / d, a.y + (a1 \* c2 - a2 \* c1) / d};

}

inline std::pair<Point, double> MCC(Point p[], int n)

{

std::random\_shuffle(p, p + n);

Point c = p[0];

double r = 0;

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

if (dist(p[i], c) > r + eps)

{

c = p[i], r = 0;

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

if (dist(p[j], c) > r + eps)

{

c = (p[i] + p[j]) / 2, r = dist(p[j], c);

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

if (dist(p[k], c) > r + eps)

c = Center(p[i], p[j], p[k]), r = dist(p[j], c);

}

}

return {c, r};

}

} // namespace geo

const int N = 2e4 + 7;

char a[N];

long long l[N];

struct PAM

{

struct node

{

int c[26];

int fail, len;

node()

{

fail = len = 0;

}

} t[N];

int num[N];

char S[N];

int n, cnt;

explicit PAM()

{

n = 0;

cnt = 1;

t[1].len = -1;

t[0].fail = 1;

}

inline int getfail(int x)

{

while (S[n] ^ S[n - t[x].len - 1]) x = t[x].fail; // 这就是为什么t[1].len要等于-1

return x;

}

inline int insert(int x)

{

S[++n] = (x -= 'a');

int cur = getfail(last);

if (!t[cur].c[x])

{

int nw = ++cnt;

t[nw].fail = t[getfail(t[cur].fail)].c[x];

t[nw].len = t[cur].len + 2;

t[cur].c[x] = nw;

num[nw] = num[t[nw].fail] + 1;

}

last = t[cur].c[x];

//++t[last].cnt;

return num[last];

}

int last;

} x;

const int N = 2e6 + 7;

namespace SAM

{

int f[N];

struct node

{

int nxt[26], len, fa;

} t[N];

#define len(x) t[x].len

#define fa(x) t[x].fa

#define nxt(x, c) t[x].nxt[c]

int node\_cnt = 1, lst = 1;

#define newnode() ++node\_cnt

inline void insert(char c)

{

int cur = newnode();

f[cur] = 1;

len(cur) = len(lst) + 1;

int p = lst;

lst = cur;

while (p && !nxt(p, c)) nxt(p, c) = cur, p = fa(p);

if (!p) return fa(cur) = 1, void();

int q = nxt(p, c);

if (len(p) + 1 == len(q)) return fa(cur) = q, void();

int clone = newnode();

t[clone] = t[q], len(clone) = len(p) + 1;

while (p && nxt(p, c) == q) nxt(p, c) = clone, p = fa(p);

fa(q) = fa(cur) = clone;

}

} // namespace SAM

using namespace SAM;

namespace KMP

{

int nxt[N];

void calc()

{

nxt[1] = 0;

for (int i = 2, j = 0; i <= m; ++i)

{

while (j && s[i] != s[j + 1]) j = nxt[j];

nxt[i] = (s[i] == s[j + 1] ? ++j : 0);

}

}

} // namespace KMP

namespace Manacher

{

int n, hw[maxn], ans;

char a[maxn], s[maxn \* 2];

void manacher()

{

int maxright = 0, mid;

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

{

if (i < maxright)

hw[i] = min(hw[maxright - i], maxright - i);

else

hw[i] = 1;

for (; s[i + hw[i]] == s[i - hw[i]]; ++hw[i])

;

if (hw[i] + i > maxright)

{

maxright = hw[i] + i;

mid = i;

}

}

}

void change()

{

s[0] = s[1] = '#';

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

{

s[i \* 2 + 2] = a[i];

s[i \* 2 + 3] = '#';

}

n = n \* 2 + 2;

s[n] = 0;

}

} // namespace Manacher

namespace ACAutomaton

{

void build()

{

for (int i = 0; i < 26; i++) ch[0][i] = 1;

Q.push(1);

nxt[1] = 0;

while (!Q.empty())

{

int u = Q.front();

Q.pop();

for (int i = 0; i < 26; ++i)

if (!ch[u][i])

ch[u][i] = ch[nxt[u]][i];

else

Q.push(ch[u][i]), nxt[ch[u][i]] = ch[nxt[u]][i];

}

}

} // namespace ACAutomaton