**ComPress**

**Index Tree**

**First element at least X**

**Tổ hợp**

**Lagrange**

**Diophantine**

**Nhân Ma trận fibo**

**POLLARD RHO algorithm**

// Usage:

// Compressor<int> comp;

// Adding an element : comp.add(value)

// After having all : comp.compress()

// Find index of value v : comp.find(v) (1-based index)

// Get the original value of index i: comp.orig[i] (1-based index)

template <class T>

struct Compressor {

vector<T> values, orig;

void add(T x) {

values.push\_back(x);

}

void compress() {

sort(values.begin(), values.end());

values.erase(unique(values.begin(), values.end()), values.end());

orig.resize(values.size() + 1);

for (int i = 0; i < values.size(); i++) {

orig[i + 1] = values[i];

}

}

// return index of value: a[i] >= x (lower\_bound)

T find(T x) {

return lower\_bound(values.begin(), values.end(), x) - values.begin() + 1;

}

// return max index: a[i] <= x

T find2(T x) {

return upper\_bound(values.begin(), values.end(), x) - values.begin();

}

};

**2Index Tree**

#include <ext/pb\_ds/assoc\_container.hpp>

using namespace \_\_gnu\_pbds;

typedef tree<int,null\_type,less<int>,rb\_tree\_tag,tree\_order\_statistics\_node\_update> IndexTree;

IndexTree s;

for (int i = 0; i <= 10; i+=2) s.insert(i);

s.erase(4);

cout << \*s.find\_by\_order(4) << endl;;

cout << s.order\_of\_key(4) << endl;;

**First element at least X**

struct SegmentTree {

struct Node {

long long maxx = 0;

};

vector<Node> st;

int n;

SegmentTree(int n): n(n) {

st.resize(4 \* n + 1);

}

void merge(Node& a, Node& b, Node& c) {

a.maxx = max(b.maxx, c.maxx);

}

void build(vector<int> &a, int id, int l, int r) {

if (l == r) {

st[id].maxx = a[l];

return;

}

int mid = (l + r) / 2;

build(a, id \* 2, l, mid);

build(a, id \* 2 + 1, mid + 1, r);

merge(st[id], st[id \* 2], st[id \* 2 + 1]);

}

void update(int id, int l, int r, int u, int val ) {

if (l == r) {

st[id].maxx = val;

return;

}

int mid = (l + r) / 2;

if (u <= mid) update(id \* 2, l, mid, u, val);

else update(id \* 2 + 1, mid + 1, r, u, val);

merge(st[id], st[id \* 2], st[id \* 2 + 1]);

}

int MaxLeast(int id, int l, int r, long long k) {

// cout << id << ' ' << l << ' ' << r << ' ' << k << endl;

if (l == r) return l;

ll mid = (l + r) / 2;

long long mx = st[id \* 2].maxx; // so luong 1 trong (l,mid)

if (mx >= k) {

// cout << "L " <<"num " << num << ' ' <<"mid " << mid << endl;

return MaxLeast(id \* 2, l, mid, k);

}

else {

// cout << "R " <<"num " << num << ' ' <<"mid " << mid << endl;

return MaxLeast(id \* 2 + 1, mid + 1, r, k);

}

}

Node query(int id, int l, int r, int u, int v) {

if (l > v || r < u) return Node();

if (u <= l && r <= v) {

return st[id];

}

int mid = (l + r) / 2;

auto a = query(id \* 2, l, mid, u, v);

auto b = query(id \* 2 + 1, mid + 1, r, u, v);

Node res;

merge(res, a, b);

return res;

}

};

vector<int> a(n + 1);

SegmentTree st(n);

st.build(a, 1, 1, n);

st.update(1, 1, n , u + 1 , v);

cout << st.MaxLeast(1,1,n,x) - 1 << endl;

Tổ hợp

long long combisub2(long long n, long long k) {

if (n < k) return 0;

return fact[n] \* inv[k]%m \* inv[n-k]%m;

}

void sub2() {

inv[0] = 1 % m;

fact[0] = 1 % m;

for (int i = 1; i <= 100000; i++) {

fact[i] = fact[i-1] \* i;

fact[i] %= m;

}

inv[100000] = pw(fact[100000],m-2);

for (int i = 100000-1; i >= 1; i--) {

inv[i] = inv[i+1] \* (i+1) % m;

}

while (t--) {

long long n,k;

cin >> n >> k;

cout << combisub2(n,k) << endl;

}

}

Lagrange

// With polynomial degree n:

// A(x) = sum (A(j) \* (mul (x - xk) / (xj - xk))) (j: 1 -> n + 1, k != j)

// 1^k + 2^k + ... + n^k -> deg = k + 1

f[0] = 0;

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

f[i] = f[i - 1] + pw(i, k);

f[i] %= MOD;

}

// degree = k, find f[n]

// O(k) - formula with i = 0 -> k

long long lagrange(vector<long long> &f, int n, int k) {

if (n <= k) return f[n];

// use prefix and suffix if n - i can be 0

long long tu = 1;

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

tu \*= (n - i);

tu %= MOD;

}

long long res = 0;

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

long long mau = (n - i) \* fact[i] % MOD \* fact[k - i] % MOD;

if ((k - i) % 2) mau \*= -1;

mau = (mau % MOD + MOD) % MOD;

res += f[i] \* tu % MOD \* pw(mau, MOD - 2) % MOD;

res %= MOD;

}

return res;

}

Diophantine

// Nghiệm: x, y

// (x, y) && (x + p, y - q): p/q = b/a (pstg)

// Họ nghiệm: (x + kp, y - kq)

struct Diophantine {

long long mulmod(long long a, long long b, long long c) {

long long sign = 1;

if (a < 0) {

a = -a;

sign = -sign;

}

if (b < 0) {

b = -b;

sign = -sign;

}

a %= c;

b %= c;

long long res = 0;

while (b > 0) {

if (b & 1) {

res = (res + a) % c;

}

a = (a + a) % c;

b >>= 1;

}

if (sign == -1) {

res = (-res) % c;

}

return res;

}

template<typename T>

T extgcd(T a, T b, T &x, T &y) {

if (a == 0) {

x = 0;

y = 1;

return b;

}

T p = b / a;

T g = extgcd(b - p \* a, a, y, x);

x -= p \* y;

return g;

}

template<typename T>

bool diophantine(T a, T b, T c, T &x, T &y, T &g) {

if (a == 0 && b == 0) {

if (c == 0) {

x = y = g = 0;

return true;

}

return false;

}

if (a == 0) {

if (c % b == 0) {

x = 0;

y = c / b;

g = abs(b);

return true;

}

return false;

}

if (b == 0) {

if (c % a == 0) {

x = c / a;

y = 0;

g = abs(a);

return true;

}

return false;

}

g = extgcd(a, b, x, y);

if (c % g != 0) {

return false;

}

T dx = c / a;

c -= dx \* a;

T dy = c / b;

c -= dy \* b;

x = dx + mulmod(x, c / g, b);

y = dy + mulmod(y, c / g, a);

g = abs(g);

return true;

}

};

// a \* k >= l

long long calcBDT(long long l, long long a) {

if (a > 0) {

if (l > 0) l = (l - 1) / a + 1;

else l = l / a;

}

else {

if (l >= 0) {

if (l % a == 0) l /= a;

else l = l / a - 1;

}

else l /= a;

}

return l;

}

int main() {

long long a, b, c;

cin >> a >> b >> c;

Diophantine diophantine;

long long x, y;

long long g = diophantine.extgcd(a, b, x, y);

cout << diophantine.diophantine(a, b, c, x, y, g) << endl;

cout << x << " " << y << " " << endl;

Nhân Ma trận Fibonaci

struct matrix{

long long a[3][3];

matrix() {

memset(a, 0, sizeof(a));

}

matrix operator \*(const matrix& b) {

matrix x;

for (int i = 1; i <= 2; i++) {

for (int j = 1; j <= 2; j++) {

for (int k = 1; k <= 2; k++) {

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

}

}

}

return x;

}

matrix operator +(const matrix& b) {

matrix x;

for (int i = 1; i <= 2; i++) {

for (int j = 1; j <= 2; j++) {

x.a[i][j] += b.a[i][j] + a[i][j];

x.a[i][j] %= MOD;

}

}

return x;

}

};

matrix donvi;

matrix pw(const matrix& A, long long c) {

if (c == 0) return donvi;

if (c == 1) return A;

matrix q = pw(A,c/2);

// cout << q.a[1][1] << ' ' << q.a[2][1] << endl;

if (c % 2 == 0) return q \* q;

else return q \* q \* (A);

}

matrix A;

matrix base;

int main() {

long long n;

cin >> n ;

base.a[1][1] = 1;

base.a[2][1] = 0;

donvi.a[2][2] = donvi.a[1][1] = 1;

donvi.a[2][1] = donvi.a[1][2] = 0;

A.a[1][1] = 1;

A.a[1][2] = 1;

A.a[2][1] = 1;

A.a[2][2] = 0;

matrix dd = pw(A,n-1)\*base;

cout << dd.a[1][1] << endl;;

// cout << base.a[1][1] << ' ' << base.a[2][1] << endl;

/\* **POLLARD RHO algorithm** \*/

// prime factorization algorithm for n = 1e18

mt19937\_64 gen(chrono::steady\_clock::now().time\_since\_epoch().count());

long long Rand(long long l, long long r){

uniform\_int\_distribution<long long> rnd(l,r);

return rnd(gen);

}

long long mul(long long a, long long b, long long m){

if (b == 1) return a % m;

if (b == 0) return 1 % m;

a %= m; b %= m;

long long q = mul(a, b / 2, m);

if (b % 2 == 0) return ((q + q) % m + m) % m;

else return ((q + q + a) % m + m) % m;

}

long long pw(long long a, long long n, long long m){

if (n == 0) return 1 % m;

long long q = pw(a, n / 2, m);

if (n % 2 == 0) return mul(q, q, m);

return mul(mul(q, q, m), a, m);

}

// Milner-Rabin algorithm

bool checkprime(long long n){

if (n == 2) return 1;

if (n % 2 == 0 || n == 1) return 0;

long long m = n - 1;

long long s = 0;

while (m % 2 == 0){

m /= 2; s++;

}

long long dem = 0, q = 0, a, b;

while (dem <= 3){

dem++;

a = Rand(2, n - 2);

b = pw(a, m, n);

if ((b + 1) % n == 0 || (b - 1) % n == 0) {q++;continue;}

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

b = mul(b, b, n);

if ((b + 1) % n == 0) {q++; break;}

}

}

if (q == 4) return 1;

return 0;

}

map<long long, long long> alpha;

void brute(int n){

if (n == 1) return;

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

if (n % i == 0) {

while (n % i == 0) {alpha[i]++; n /= i;}

}

}

if (n > 1){

alpha[n]++;

}

}

long long f(long long x, long long m){

return (mul(x, x, m) + 1) % m;

}

long long findFactor(long long n){

long long x = Rand(2, n - 1);

long long y = x;

long long p = 1;

while (p == 1){

x = f(x, n);

y = f(f(y, n), n);

p = \_\_gcd(abs(x - y), n);

}

return p;

}

void fact(long long n){

if (n == 1) return ;

if (checkprime(n)) {alpha[n]++; return;}

if (n <= 10000) {

brute(n); return;

}

long long p = 0;

while (p == 0 || p == n){

p = findFactor(n);

}

fact(p); fact(n / p);

}

// find all divisor

vector<long long>ts;

void backtrack(int val, vector<pair<long long,int>>&a, long long res) {

if (val == a.size()) {

ts.pb(res);

// cout << res << endl;

return;

}

backtrack(val + 1, a, res);

for (int i = 1; i <= a[val].se; i++) {

res = res \* a[val].fi;

// cout << i << ' ' << a[val] << ' ' << cnt[a[val]] << ' ' << res << endl;

backtrack(val + 1, a, res);

}

}

int main() {

long long n; cin >> n;

alpha.clear();

fact(n);

vector<pair<long long,int>>a;

for (auto i: alpha){

a.pb({i.first,i.second});

cout << i.first << "^" << i.second << "\n";

}

// find all divisor

backtrack(0,a,1);

sort(All(ts));

for (auto i : ts) cout << i << ' ';

cout << endl;

ts.clear();

//

val.clear();