Mục lục

[1. Dijkstra 1](#_Toc147591928)

[2. Hash 1](#_Toc147591929)

[3. LCA 1](#_Toc147591930)

[4. Number theory 1](#_Toc147591931)

[5. Prim 1](#_Toc147591932)

[6. Lazy SegTree 1](#_Toc147591933)

[7. KMP 1](#_Toc147591934)

[8. Suffix array 1](#_Toc147591935)

[9. Convex hull and some geometry 1](#_Toc147591936)

[10. Ternary search 1](#_Toc147591937)

21. FastIO ………………………………………………………………………..……18

22. Graph………………………………………………………………………..……..21

23. Segment Tree…………………………………………………………………..23

24. Prime Sieve………………………………………………………………………24

25. Hash………………………………………………………………………………24

26. DisjointSet Union………………………………………………………………..25

1. Dijkstra

const long long INF = 1e18;

int n, m;

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

vector<long long> dis, trace;

void dijkstra(int st) {

priority\_queue<pair<long long, int>, vector<pair<long long, int>>, greater<pair<long long, int>>> pq;

trace.assign(n + 1, -1);

dis.assign(n + 1, INF);

dis[st] = 0, trace[st] = st;

pq.push(make\_pair(0, st));

while (pq.size()) {

int u = pq.top().second;

long long du = pq.top().first;

pq.pop();

if (du != dis[u])

continue;

for (int i = 0; i < c[u].size(); i++) {

int v = c[u][i].first;

long long uv = c[u][i].second;

if (dis[v] > du + uv) {

dis[v] = du + uv, trace[v] = u;

pq.push(make\_pair(dis[v], v));

}

}

}

}

vector<int> getPath(int fi) {

int u = fi;

vector<int> path;

path.push\_back(u);

while (u != trace[u] && trace[u] != -1) {

u = trace[u];

path.push\_back(u);

}

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

return path;

}

int main() {

int u, v, w;

cin >> n >> m;

c.resize(n + 1);

while (m--) {

cin >> u >> v >> w;

c[u].push\_back({v, w});

c[v].push\_back({u, w});

}

dijkstra(1);

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

cout << dis[i] << ": ";

vector<int> path = getPath(i);

for (int j = 0; j < path.size(); j++)

cout << path[j] << ' ';

}

1. Hash

// const long long BASE = 10042003;

// const long long MOD = 1e9 + 7;

// xóa BASE, MOD ở dòng 18 khi TLE

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

if (n == 0) return 1;

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

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

return q \* q % MOD \* a % MOD;

}

struct HashString {

string str;

vector<long long> prefixSum, inv;

long long MOD, BASE;

// constructor

HashString(string str = "", long long BASE = 10042003, long long MOD = 1e9 + 7):

str(str), MOD(MOD), BASE(BASE) {

int n = str.size();

prefixSum.resize(n + 1);

inv.resize(n + 1);

// calc inv

long long invBase = pw(BASE, MOD - 2, MOD);

inv[0] = 1;

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

inv[i] = inv[i - 1] \* invBase % MOD;

}

// calc prefixSum = sum s[i] \* 10^i

long long curPow = 1; // BASE ^ i

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

prefixSum[i] = (prefixSum[i - 1] + curPow \* str[i - 1] % MOD) % MOD;

curPow = curPow \* BASE % MOD;

}

}

// return hash of str[l..r]

long long query(int l, int r) {

long long res = (prefixSum[r] - prefixSum[l - 1] + MOD) % MOD;

res = res \* inv[l] % MOD; // res /= pw(BASE, l)

return res;

}

};

int main() {

string s;

cin >> s;

HashString hash(s, 4599, 1e9 + 9);

cout << hash.MOD << endl;

hash.MOD = 103;

cout << hash.MOD << endl;

vector<HashString> hash(n + 1, HashString(""));

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

string s;

cin >> s;

hash[i] = HashString(s);

}

1. LCA

//2^max\_h >= n

const int max\_h = 20;

vector<vector<int>> c;

vector<vector<int>> up;

vector<int> depth;

void buildBL(int u, int par) {

up[u][0] = par;

for (int i = 1; i < max\_h; i++)

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

for (int v : c[u]) {

if (v == par)

continue;

depth[v] = depth[u] + 1;

buildBL(v, u);

}

}

int upByK(int u, int k) {

bitset<max\_h> bs(k);

for (int i = 0; i < max\_h; i++) {

if (bs[i])

u = up[u][i];

}

return u;

}

int getLCA(int u, int v) {

if (depth[u] < depth[v])

swap(u, v);

int depth\_diff = depth[u] - depth[v];

u = upByK(u, depth\_diff);

for (int i = max\_h - 1; i >= 0; i--) {

if (up[u][i] != up[v][i]) {

u = up[u][i];

v = up[v][i];

}

}

if (u != v)

u = up[u][0];

return u;

}

//make long long if need

int dist(int u, int v) {

int lca = getLCA(u, v);

return depth[u] + depth[v] - 2 \* depth[lca];

}

int main() {

int n, q, a, b;

cin >> n >> q;

c.resize(n);

depth.assign(n, 0);

up.assign(n, vector<int>(max\_h));

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

cin >> a;

c[i].push\_back(a);

c[a].push\_back(i);

}

buildBL(0, 0);

while (q--) {

cin >> a >> b;

cout << getLCA(a, b) << '\n';

}

return 0;

}

1. Number theory

long long modExpo(long long x, long long n, long long M){

if(n == 0)

return 1;

else if(n % 2 == 0)

return modExpo((x \* x) % M, n / 2, M);

else

return (x \* modExpo((x \* x) % M, (n - 1) / 2, M)) % M;

}

int d,x,y;

void extEuclid(int A, int B) {

if(B == 0) {

d = A;

x = 1;

y = 0;

} else {

extEuclid(B, A % B);

int temp = x;

x = y;

y = temp - (A / B) \* y;

}

}

//gcd(A, M) == 1

int modInvEE(int A, int M) {

extEuclid(A, M);

return (x % M + M) % M;

}

//isPrime(M) == true

int modInvFE(int A,int M) {

return modExpo(A,M-2,M);

}

Sieve

int minPrime[n + 1];

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

if (minPrime[i] == 0) { //if i is prime

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

if (minPrime[j] == 0) {

minPrime[j] = i;

}

}

}

}

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

if (minPrime[i] == 0) {

minPrime[i] = i;

}

}

void rangeSieve(int L, int R, vector<bool> &isPrime) {

isPrime.assign(R - L + 1, true);

for (long long i = 2; i \* i <= R; ++i) {

for (long long j = max(i \* i, (L + i - 1) / i \* i); j <= R; j += i) {

isPrime[j - L] = false;

}

}

if (1 >= L) {

isPrime[1 - L] = false;

}

}

int xorOneToN(int n) {

if (n % 4 == 0)

return n;

if (n % 4 == 1)

return 1;

if (n % 4 == 2)

return n + 1;

return 0;

}

int main() {

cout << modExpo(2, 1000, 1e9 + 7) << endl;

cout << modInvEE(2, 1e9 + 7) << endl;

cout << modInvFE(2, 1e9 + 7) << endl;

vector<bool> isPrime;

sieve(1e6, isPrime);

for (int i = 0; i <= 10; i++) {

if (isPrime[i])

cout << i << ' ';

}

cout << endl;

vector<int> minPrime, facs;

facSieve(1e6, minPrime);

facs = factorize(100, minPrime);

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

cout << facs[i] << ' ';

cout << endl;

vector<bool> rgSieve;

int L = 1e7, R = 1e7 + 1e4;

rangeSieve(L, R, rgSieve);

for (long long x = L; x <= R; ++x) {

if (rgSieve[x - L]) {

cout << x << ' ';

}

}

cout << endl;

return 0;

}

1. Prim

const int INF = 1e18;

int n, m;

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

vector<long long> dis;

long long prim(int s) {

long long ret = 0;

priority\_queue<pair<long long, int>, vector<pair<long long, int>>, greater<pair<long long, int>>> q;

fill(dis.begin(), dis.end(), INF);

dis[s] = 0;

q.push({0, s});

while (!q.empty()) {

auto top = q.top(); q.pop();

long long curDis = top.first; int u = top.second;

if (curDis != dis[u]) continue;

ret += dis[u]; dis[u] = -INF;

for (auto &e : c[u]) {

int v = e.first; long long cc = e.second;

if (dis[v] > cc) {

dis[v] = cc;

q.push({dis[v], v});

}

}

}

return ret;

}

int main() {

cin >> n >> m;

c.clear();

c.resize(n + 1);

dis.assign(n + 1, INF);

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

int u, v, cc;

cin >> u >> v >> cc;

c[u].push\_back({v, cc});

c[v].push\_back({u, cc});

}

cout << prim(1) << '\n';

}

1. Lazy SegTree

struct SegmentTree {

struct Node {

long long sum = 0;

long long lazy = -1; // -1 nếu update là phép gán

};

vector<Node> st;

int n;

SegmentTree(int n): n(n) {

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

}

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

a.sum = b.sum + c.sum;

}

void build(int id, int l, int r) {

if (l == r) {

st[id].sum = 0;

return;

}

int mid = (l + r) / 2;

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

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

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

}

void down(int id, int l, int r) {

if (l == r || st[id].lazy == -1) return; // st[id].lazy == Node().lazy

long long val = st[id].lazy;

int mid = (l + r) / 2;

st[id \* 2].sum = val \* (mid - l + 1);

st[id \* 2 + 1].sum = val \* (r - mid);

st[id \* 2].lazy = val;

st[id \* 2 + 1].lazy = val;

st[id].lazy = -1;

}

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

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

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

st[id].sum = val \* 1ll \* (r - l + 1);

st[id].lazy = val;

return;

}

down(id, l, r);

int mid = (l + r) / 2;

update(id \* 2, l, mid, u, v, val);

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

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

}

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

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

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

return st[id];

}

down(id, l, r);

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;

}

};

SegmentTree st(n + 1);

st.build(1,1,n);

st.update(1,1,n,l,r,val);

cout << st.query(1,1,n,l,r).sum << endl;

1. KMP

vector<int> prefix\_function(string s) {

int n = (int)s.length();

vector<int> pi(n);

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

int j = pi[i-1];

while (j > 0 && s[i] != s[j])

j = pi[j-1];

if (s[i] == s[j])

j++;

pi[i] = j;

}

return pi;

}

1. Suffix array

// sorted suffix, ans[i] = x => suffix start at x

vector<int> sort\_cyclic\_shifts(string const& s) {

int n = s.size();

const int alphabet = 256;

vector<int> p(n), c(n), cnt(max(alphabet, n), 0);

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

cnt[s[i]]++;

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

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

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

p[--cnt[s[i]]] = i;

c[p[0]] = 0;

int classes = 1;

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

if (s[p[i]] != s[p[i-1]])

classes++;

c[p[i]] = classes - 1;

}

1. Convex hull and some geometry

struct Point {

long long x, y;

Point (long long x = 0, long long y = 0): x(x), y(y) {}

Point operator + (const Point &a) const{

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

}

Point operator - (const Point &a) const{

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

}

long long operator \* (const Point &a) const{

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

}

bool operator < (const Point &a) {

if (x == a.x) return y < a.y;

return x < a.x;

}

bool operator == (const Point &a) {

return (x == a.x && y == a.y);

}

};

// cross product

long long cross(const Point &a, const Point &b) {

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

}

long long cross2(const Point a, const Point b) {

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

}

// area of triangle

long long area(Point a, Point b, Point c) {

return cross(a, b) + cross(b, c) + cross(c, a); // area\*2, abs(res)

}

vector<Point> convexHull(vector<Point> pts) {

int n = pts.size();

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

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

vector<Point> up, dn;

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

// Note: If need maximum points on convex hull, need to change >= 0 and <= 0 to > 0 and < 0.

while (up.size() > 1 && area(up[up.size()-2], up.back(), pts[i]) >= 0) up.pop\_back();

while (dn.size() > 1 && area(dn[dn.size()-2], dn.back(), pts[i]) <= 0) dn.pop\_back();

up.push\_back(pts[i]);

dn.push\_back(pts[i]);

}

pts = dn;

for (int i = (int) up.size() - 2; i >= 1; i--) pts.push\_back(up[i]);

if (pts.size() > n) pts = {dn[0], dn.back()}; // pts = dn if need maximum points

return pts;

}

// area of polygon

long long areaPolygon(vector<Point> &a) {

long long res = 0;

int n = a.size();

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

res += cross2(a[i], a[i - 1]);

}

res += cross2(a[0], a[n - 1]);

return abs(res); // 2 \* area

}

int main() {

int n;

cin >> n;

vector<Point> a(n);

for (int i = 0; i < n; i++){ cin >> a[i].x >> a[i].y;

auto p = convexHull(a);

// for (auto i: p) {

// cout << i.x << ' ' << i.y << endl;

// }

long long res = areaPolygon(p);

1. Ternary search

double max\_f(double left, double right) {

int N\_ITER = 100;

for (int i = 0; i < N\_ITER; i++) {

double x1 = left + (right - left) / 3.0;

double x2 = right - (right - left) / 3.0;

if (f(x1) > f(x2)) right = x2;

else left = x1;

}

return f(left);

}

1. Persistent Segment Tree

struct Node {

int left, right; // ID of left child & right child

long long ln; // Max value of node

Node() {}

Node(long long ln, int left, int right) : ln(ln), left(left), right(right) {}

} it[11000111]; // Each node has a position in this array, called ID

int nNode;

int ver[MN]; // ID of root in each version

// Update max value of a node

inline void refine(int cur) {

it[cur].ln = max(it[it[cur].left].ln, it[it[cur].right].ln);

}

// Update a range, and return new ID of node

int update(int l, int r, int u, int x, int oldId) {

if (l == r) {

++nNode;

it[nNode] = Node(x, 0, 0);

return nNode;

}

int mid = (l + r) >> 1;

int cur = ++nNode;

if (u <= mid) {

it[cur].left = update(l, mid, u, x, it[oldId].left);

it[cur].right = it[oldId].right;

refine(cur);

}

else {

it[cur].left = it[oldId].left;

it[cur].right = update(mid+1, r, u, x, it[oldId].right);

refine(cur);

}

return cur;

}

// Get max of range. Same as usual IT

int get(int nodeId, int l, int r, int u, int v) {

if (v < l || r < u) return -1;

if (u <= l && r <= v) return it[nodeId].ln;

int mid = (l + r) >> 1;

return max(get(it[nodeId].left, l, mid, u, v), get(it[nodeId].right, mid+1, r, u, v));

}

// When update:

++nVer;

ver[nVer] = update(1, n, u, x, ver[nVer-1]);

// When query:

res = get(ver[t], 1, n, u, v);

1. DSU

long long root[200005];

vector<int>adj[200005];

long long add[200005];

long long res[200005];

int findRoot(int u) {

while (root[u] >= 0) {

u = root[u];

}

return u;

}

void merge(int a, int b) {

a = findRoot(a);

b = findRoot(b);

if (a == b) return;

if (root[a] > root[b]) {

swap(a, b);

}

root[a] += root[b];

root[b] = a;

adj[a].pb(b);

res[b] = res[b] + add[b] - add[a];

for (auto i : adj[b]) {

adj[a].pb(i);

res[i] = res[i] + add[b] - add[a];

}

add[b] = 0;

adj[b].clear();

return;

}

long long get(int a) {

return add[findRoot(a)] + res[a];

}

void update(int u, int point) {

u = findRoot(u);

add[u] += point;

return;

}

1. Compressor

// 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();

}

};

1. **Index 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;;

1. **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) {

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

}

else {

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;

1. 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;

}

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;

}

long long n,k;

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

1. 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;

}

1. 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;

1. 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;

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;

1. POLLARD RHO

// 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();

22. Theorem:

• Số ước của n=p1^x\*p2\*y.. với p1, p2.. là ước nguyên tố của n => số ước bằng (x+1)\*(y+1)..

• Từ 0 tới n có xấp xỉ n / ln(n) số nguyên tố

• Cách tính số lượng số nguyên tố cùng nhau với n và nhỏ hơn bằng n: n = p1^x\*p2^y.. => đáp án là: p1^(x-1)\*(p1-1)\*p2^(y-1)\*(p2-1)..

• Nim game: Có n hộp, mỗi hộp chứa 1 số lượng sỏi, người chơi lần lượt bốc sỏi trong những hộp còn sỏi, ai không đi được thì thua. VD có n hộp với số lượng sỏi x1,x2..xn. Gọi s = x1^x2^..^xn. Nếu s = 0 thì thua ngược lại thắng.

• Chia x viên sỏi vào y hộp => (y+x-1)C(y-1)

• Số lương dãy ngược đúng độ dài 2\*n: C(n) = (2\*nCn)/(n+1). Số lương cây nhị phân có n nút là C(n), số lương cây n nút là C(n-1)

Cho 2 số nguyên a,p nguyên tố cùng nhau, p là 1 số nguyên tố, ta có được tính chất sau: a^{p – 1 đồng dư với 1 (mod p)