**MInt**

template<class T>

constexpr T power(T a, LL b) {

T res = 1;

for (; b; b /= 2, a \*= a) {

if (b % 2) {

res \*= a;

}

}

return res;

}

template<int P>

struct MInt {

int x;

constexpr MInt() : x{} {}

constexpr MInt(LL x) : x{norm(x % getMod())} {}

static int Mod;

constexpr static int getMod() {

if (P > 0) {

return P;

} else {

return Mod;

}

}

constexpr static void setMod(int Mod\_) {

Mod = Mod\_;

}

constexpr int norm(int x) const {

if (x < 0) {

x += getMod();

}

if (x >= getMod()) {

x -= getMod();

}

return x;

}

constexpr int val() const {

return x;

}

explicit constexpr operator int() const {

return x;

}

constexpr MInt operator-() const {

MInt res;

res.x = norm(getMod() - x);

return res;

}

constexpr MInt inv() const {

assert(x != 0);

return power(\*this, getMod() - 2);

}

constexpr MInt &operator\*=(MInt rhs) & {

x = 1LL \* x \* rhs.x % getMod();

return \*this;

}

constexpr MInt &operator+=(MInt rhs) & {

x = norm(x + rhs.x);

return \*this;

}

constexpr MInt &operator-=(MInt rhs) & {

x = norm(x - rhs.x);

return \*this;

}

constexpr MInt &operator/=(MInt rhs) & {

return \*this \*= rhs.inv();

}

friend constexpr MInt operator\*(MInt lhs, MInt rhs) {

MInt res = lhs;

res \*= rhs;

return res;

}

friend constexpr MInt operator+(MInt lhs, MInt rhs) {

MInt res = lhs;

res += rhs;

return res;

}

friend constexpr MInt operator-(MInt lhs, MInt rhs) {

MInt res = lhs;

res -= rhs;

return res;

}

friend constexpr MInt operator/(MInt lhs, MInt rhs) {

MInt res = lhs;

res /= rhs;

return res;

}

friend constexpr istream &operator>>(std::istream &is, MInt &a) {

LL v;

is >> v;

a = MInt(v);

return is;

}

friend constexpr ostream &operator<<(ostream &os, const MInt &a) {

return os << a.val();

}

friend constexpr bool operator==(MInt lhs, MInt rhs) {

return lhs.val() == rhs.val();

}

friend constexpr bool operator!=(MInt lhs, MInt rhs) {

return lhs.val() != rhs.val();

}

};

template<>

int MInt<0>::Mod = 998244353;

template<int V, int P>

constexpr MInt<P> CInv = MInt<P>(V).inv();

constexpr int P = 1000000007;

using Z = MInt<P>;

**并查集**

struct DSU {

std::vector<int> f, siz;

DSU(int n) : f(n + 1), siz(n + 1, 1) {

std::iota(f.begin(), f.end(), 0);

}

int find(int x) {

while (x != f[x]) x = f[x] = f[f[x]];

return x;

}

bool same(int x, int y) {

return find(x) == find(y);

}

bool merge(int x, int y) {

x = find(x);

y = find(y);

if (x == y) return false;

siz[x] += siz[y];

f[y] = x;

return true;

}

int size(int x) {

return siz[find(x)];

}

};

**一维树状数组**

template <typename T>

struct fenwick {

int n;

std::vector<T> tr;

fenwick(int \_n = 0) {

n = \_n + 1;

tr.assign(n, T{});

}

void add(int x, const T &v) {

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

tr[i - 1] = tr[i - 1] + v;

}

}

T Sum(int x) {

T ans{};

for (int i = x + 1; i > 0; i -= i & -i) {

ans = ans + tr[i - 1];

}

return ans;

}

T Sum(int l, int r) {

return Sum(r) - Sum(l - 1);

}

int kth(const T &k) {

int x = 0;

T cur{};

for (int i = 1 << std::\_\_lg(n); i; i /= 2) {

if (x + i <= n && cur + tr[x + i - 1] < k) {

x += i;

cur = cur + tr[x - 1];

}

}

return x;

}

};

**二维树状数组**

template<typename T>

struct fenwick2D{

vector<vector<T>> tr;

int n, m;

fenwick2D(int \_n, int \_m) : n(\_n), m(\_m) {

tr.resize(n + 1);

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

tr.resize(m + 1);

}

}

void add(int x, int y, int val){ // 1 <= x <= n, 1 <= y <= m

for(int i = x; i <= n; i += i & -i) {

for(int j = y; j <= m; j += j & -j) {

tr[i][j] += val;

}

}

}

T Sum(int x, int y) { // 1 <= x <= n, 1 <= y <= m

T res = 0;

for(int i = x; i > 0; i -= i & -i) {

for(int j = y; j > 0; j -= j & -j) {

res += tr[i][j];

}

}

return res;

}

T Sum(int x1, int y1, int x2, int y2) { // sum[x1..x2, y1..y2]

return Sum(x2, y2) - Sum(x2, y1 - 1) - Sum(x1 - 1, y2) + Sum(x1 - 1, y1 - 1);

}

};

**区间修改，单点询问（树状数组版）**

template <typename T>

struct fenwick {

int n;

std::vector<std::vector<T>> tr;

fenwick(int \_n) : n(\_n) {

tr.resize(2);

for (int i = 0; i < 2; i++) {

tr[i].resize(n + 2);

}

}

void add(int i, int x, const T &v) {

for ( ; x <= n; x += x & -x) {

tr[i][x] += v;

}

}

void modify(int l, int r, const T &v) {

add(0, l, v);

add(0, r + 1, -v);

add(1, l, l \* v);

add(1, r + 1, (r + 1) \* (-v));

}

T sum(int i, int x) {

T ans = 0;

for ( ; x > 0; x -= x & -x) {

ans += tr[i][x];

}

return ans;

}

T Sum(int x) {

return sum(0, x) \* (x + 1) - sum(1, x);

}

T Sum(int l, int r) {

return Sum(r) - Sum(l - 1);

}

};

**动态中位数（对顶堆）**

struct DynamicMedian {

priority\_queue<int> down;

priority\_queue<int, vector<int>, greater<int>> up;

DynamicMedian() {}

void insert(int x) {

if (down.empty() || x <= down.top()) {

down.push(x);

} else {

up.push(x);

}

if (down.size() > 1 + up.size()) {

up.push(down.top());

down.pop();

}

if (up.size() > down.size()) {

down.push(up.top());

up.pop();

}

};

double Ans() {

if (up.size() + down.size() & 1) {

return down.top();

} else {

return (down.top() + up.top()) / 2.0;

}

};

};

**线段树**

template<class Info, class Tag>

struct LazySegmentTree {

int n;

std::vector<Info> info;

std::vector<Tag> tag;

LazySegmentTree() : n(0) {}

LazySegmentTree(int n\_, Info v\_ = Info()) {

init(n\_, v\_);

}

template<class T>

LazySegmentTree(std::vector<T> init\_) {

init(init\_);

}

void init(int n\_, Info v\_ = Info()) {

init(std::vector(n\_ + 1, v\_));

}

template<class T>

void init(std::vector<T> init\_) {

n = init\_.size() - 1;

info.assign(4 << std::\_\_lg(n + 1), Info());

tag.assign(4 << std::\_\_lg(n + 1), Tag());

auto build = [&](auto build, int u, int l, int r) -> void {

if (l == r) {

info[u] = {init\_[l]};

return;

}

int mid = l + r >> 1;

build(build, u << 1, l, mid);

build(build, u << 1 | 1, mid + 1, r);

pushup(u);

};

build(build, 1, 0, n);

}

void pushup(int u) {

info[u] = info[u << 1] + info[u << 1 | 1];

}

void apply(int u, const Tag &v) {

info[u].apply(v);

tag[u].apply(v);

}

void pushdown(int u) {

apply(u << 1, tag[u]);

apply(u << 1 | 1, tag[u]);

tag[u] = Tag();

}

void modify(int u, int l, int r, int x, const Info &v) {

if (l == r) {

info[u] = v;

return;

}

int mid = l + r >> 1;

pushdown(u);

if (x <= mid) {

modify(u << 1, l, mid, x, v);

} else {

modify(u << 1 | 1, mid + 1, r, x, v);

}

pushup(u);

}

void modify(int p, const Info &v) {

modify(1, 0, n, p, v);

}

void rangeApply(int u, int l, int r, int x, int y, const Tag &v) {

if (r < x || l > y) {

return;

}

if (l >= x && r <= y) {

apply(u, v);

return;

}

int mid = l + r >> 1;

pushdown(u);

rangeApply(u << 1, l, mid, x, y, v);

rangeApply(u << 1 | 1, mid + 1, r, x, y, v);

pushup(u);

}

void Apply(int p, const Tag &v) {

rangeApply(1, 0, n, p, p, v);

}

void rangeApply(int l, int r, const Tag &v) {

rangeApply(1, 0, n, l, r, v);

}

Info rangeQuery(int u, int l, int r, int x, int y) {

if (r < x || l > y) {

return Info();

}

if (x <= l && r <= y) {

return info[u];

}

int mid = l + r >> 1;

pushdown(u);

if (y <= mid) {

return rangeQuery(u << 1, l, mid, x, y);

} else if (x > mid) {

return rangeQuery(u << 1 | 1, mid + 1, r, x, y);

}

auto left = rangeQuery(u << 1, l, mid, x, y);

auto right = rangeQuery(u << 1 | 1, mid + 1, r, x, y);

return left + right;

}

Info Query(int p) {

return rangeQuery(1, 0, n, p, p);

}

Info rangeQuery(int l, int r) {

return rangeQuery(1, 0, n, l, r);

}

};

struct Tag {

void apply(const Tag &t) {

}

};

struct Info {

void apply(const Tag &t) {

}

};

Info operator+(const Info &a, const Info &b) {

}

**RMQ**

template <class Info>

struct RMQ {

std::vector<int> lg;

std::vector<std::vector<Info>> f;

RMQ(std::vector<int> a) : lg(a.size()) {

int n = a.size() - 1, m = \_\_lg(n) + 1;

f.resize(n + 1);

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

f[i].resize(m + 1);

f[i][0] = {a[i]};

if (i) lg[i] = \_\_lg(i);

}

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

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

f[i][j] = f[i][j - 1] + f[i + (1 << j - 1)][j - 1];

}

}

}

Info Query(int l, int r) {

int k = lg[r - l + 1];

return f[l][k] + f[r - (1 << k) + 1][k];

}

};

struct Info {

int x;

friend Info operator+(const Info &a, const Info &b) {

}

};

**ST表**

template <class Info>

struct ST {

int n, m;

std::vector<int> lg;

std::vector<std::vector<Info>> f;

ST(std::vector<int> a) : n(a.size() - 1), lg(a.size()) {

m = \_\_lg(n) + 1;

f.resize(n + 1);

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

f[i].resize(m + 1);

f[i][0] = {a[i]};

if (i) lg[i] = \_\_lg(i);

}

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

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

f[i][j] = f[i][j - 1] + f[i + (1 << j - 1)][j - 1];

}

}

}

Info Query(int l, int r) {

Info ans = f[l][0];

int p = l + 1;

for (int i = m; i >= 0; i--) {

if (p + (1 << i) - 1 <= r) {

ans = ans + f[p][i];

p += 1 << i;

}

}

return ans;

}

};

struct Info {

int x;

friend Info operator+(const Info &a, const Info &b) {

}

};

**单哈希**

/\*

const int M = 9'223'372'036'854'775'783;

\*/

template <typename T, const int P>

struct single\_hash {

int n;

std::vector<T> h, hi, p;

single\_hash(string s) : n(s.size() - 1), h(n + 1), hi(n + 2), p(n + 1) {

p[0] = 1;

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

p[i] = p[i - 1] \* P;

h[i] = h[i - 1] \* P + s[i];

}

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

hi[i] = hi[i + 1] \* P + s[i];

}

}

T get(int l, int r) {

return h[r] - h[l - 1] \* p[r - l + 1];

}

T geti(int l, int r) {

return hi[l] - hi[r + 1] \* p[r - l + 1];

}

bool ispalindrome(int l, int r) {

return get(l, r) == geti(l, r);

}

bool same(int l1, int r1, int l2, int r2) {

return get(l1, r1) == get(l2, r2);

}

T MergeFF(int l1, int r1, int l2, int r2) { // Forward and Forward

return get(l1, r1) \* p[r2 - l2 + 1] + get(l2, r2);

}

T MergeFR(int l1, int r1, int l2, int r2) { // Forward and reverse

return get(l1, r1) \* p[r2 - l2 + 1] + geti(l2, r2);

}

T MergeRF(int l1, int r1, int l2, int r2) { // reverse and Forward

return geti(l1, r1) \* p[r2 - l2 + 1] + get(l2, r2);

}

T MergeRR(int l1, int r1, int l2, int r2) { // reverse and reverse

return geti(l1, r1) \* p[r2 - l2 + 1] + geti(l2, r2);

}

};

using SingleHash = single\_hash<MInt<1410412741>, 131>;

**珂朵莉树**

struct ODT {

struct odt {

int l, r;

mutable int x;

bool operator < (const odt &a) const {

return l < a.l;

}

};

set<odt> tr;

typedef set <odt> :: iterator IT;

ODT(int l, int r, int x) {

tr.insert({l, r, x});

}

IT split(int pos) { //将pos-1和pos之间切开，返回pos所在区间指针

auto it = tr.lower\_bound({pos, 0, 0});

if (it != tr.end() && it->l == pos) return it;

it--;

int l = it->l, r = it->r, x = it->x;

tr.erase(it);

tr.insert({l, pos - 1, x});

return tr.insert({pos, r, x}).first;

}

void assign(int l, int r, int x) {

auto R = split(r + 1);

auto L = split(l);

tr.erase(L, R);

tr.insert({l, r, x});

}

void modify(int l, int r) {

auto R = split(r + 1);

auto L = split(l);

for (auto it = L; it != R; it++) {

// 对it->x暴力修改

}

}

int query() {

int ans = 0;

for (auto it = tr.begin(); it != tr.end(); it++) {

}

return ans;

}

};

**组合数**

namespace Combinatorics {

int n;

std::vector<Z> \_fac = {1}, \_inv = {1};

void init(int m) {

if (m <= n) return;

\_fac.resize(m + 1);

\_inv.resize(m + 1);

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

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

}

\_inv[m] = \_fac[m].inv();

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

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

}

n = m;

}

Z fac(int m) {

if (m > n) init(2 \* m);

return \_fac[m];

}

Z inv(int m) {

if (m > n) init(2 \* m);

return \_inv[m];

}

Z C(int a, int b) {

if (a < b || b < 0) return Z(0);

return fac(a) \* inv(b) \* inv(a - b);

}

Z A(int a, int b) {

if (a < b || b < 0) return Z(0);

return fac(a) \* inv(a - b);

}

Z H(int a, int b) {

return C(a + b - 1, b);

}

}

using namespace Combinatorics;

**扩展欧几里得求逆元**

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

if (!b) {

x = 1, y = 0;

return a;

}

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

y -= a / b \* x;

return d;

}

int inv(int n, int M) {

// n \* x = 1 (mod M) -> n \* x + M \* y = 1

int x, y;

exgcd(n, M, x, y);

x = (x % M + M) % M;

return x;

}

**扩展欧几里得求解不定方程**

求解 a \* x + b \* y = c

a, b, c 已知，求出满足条件的一对x, y

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

if (c % d == 0) {

x \*= c / d;

y \*= c / d;

int dx = abs(b / d);

int dy = abs(a / d);

} else {

// 无解

}

通过以上代码即可求出

dx, dy是满足等式两边平衡的x, y的最小变化量

**求斐波那契数列第n项**

struct matrix {

Z x[2][2];

friend matrix operator \*(const matrix &lhs, const matrix &rhs) {

matrix res = {0, 0, 0, 0};

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

for (int k = 0; k < 2; k++) {

res.x[i][j] += lhs.x[i][k] \* rhs.x[k][j];

}

}

}

return res;

}

};

Z Fib(int n) {

if (!n) return 0;

matrix a = {0, 1, 1, 1}, res = {1, 0, 0, 1};

int k = n - 1;

while (k) {

if (k & 1) res = res \* a;

a = a \* a;

k >>= 1;

}

return res.x[0][0] + res.x[1][0];

}

**素数打表**

namespace PrimeTable {

std::vector<int> primes;

std::vector<int> minp;

std::vector<bool> is\_prime;

void init(int n) {

minp.resize(n + 1);

is\_prime.resize(n + 1, true);

is\_prime[0] = is\_prime[1] = false;

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

if (is\_prime[i]) {

primes.push\_back(i);

minp[i] = i;

}

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

is\_prime[primes[j] \* i] = false;

minp[primes[j] \* i] = primes[j];

if(i % primes[j] == 0) break;

}

}

}

bool IsPrime(int n) {

if (n <= 1) return false;

return is\_prime[n];

}

}

using namespace PrimeTable;

**Miller\_Rabin大素数判定**

std::ostream &operator<<(std::ostream &os, \_\_int128 n) {

std::string s;

while (n) {

s += '0' + n % 10;

n /= 10;

}

std::reverse(s.begin(), s.end());

return os << s;

}

\_\_int128 mul(\_\_int128 a, \_\_int128 b, \_\_int128 M) {

a = (a % M + M) % M;

b = (b % M + M) % M;

\_\_int128 ans = 0;

while (b) {

if (b & 1) {

ans += a;

if (ans >= M) {

ans -= M;

}

}

a <<= 1;

if (a >= M) {

a -= M;

}

b >>= 1;

}

return ans;

}

\_\_int128 power(\_\_int128 n, \_\_int128 k, \_\_int128 M) {

\_\_int128 ans = 1;

while (k) {

if (k & 1) {

ans = mul(ans, n, M);

}

n = mul(n, n, M);

k >>= 1;

}

return ans;

}

bool check(\_\_int128 x,\_\_int128 M){

\_\_int128 mid = power(x, M - 1, M);

if (mid != 1) return false;

\_\_int128 n = M - 1;

while (n % 2 == 0 && mid == 1) {

n >>= 1;

mid = power(x, n, M);

}

if (mid == 1 || mid == M - 1) return true;

return false;

}

bool Miller\_Rabin(LL x) {

vector<int> prime = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47};

if (x <= 47) {

for (int i = 0; i < 15; i++) {

if (prime[i] == x) {

return true;

}

}

return false;

}

for (int i = 0; i < 15; i++) {

if (!check(prime[i], x)) {

return false;

}

}

return true;

}

**tarjan缩点**

struct tarjan {

int timestamp;

vector<int> dfn, low, id;

vector<vector<array<int, 2>>> g;

stack<int> st;

vector<bool> is\_bridge;

vector<vector<int>> dcc;

tarjan(int n, int m, vector<int> &x, vector<int> &y) {

g.resize(n + 1);

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

g[x[i]].push\_back({y[i], i});

g[y[i]].push\_back({x[i], i});

}

dfn.resize(n + 1);

low.resize(n + 1);

timestamp = 0;

id.resize(n + 1);

is\_bridge.resize(m + 1);

dfs(1, -1);

}

void dfs(int u, int fa) {

dfn[u] = low[u] = ++timestamp;

st.push(u);

for (auto &[v, i] : g[u]) {

if (v == fa) continue;

if (!dfn[v]) {

dfs(v, u);

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

if(low[v] > dfn[u]) {

is\_bridge[i] = true;

}

} else if (dfn[v] < dfn[u]) {

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

}

}

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

vector<int> t;

do {

t.push\_back(st.top());

st.pop();

id[t.back()] = dcc.size() + 1;

} while (t.back() != u);

dcc.push\_back(t);

}

}

};

**求 l & (l + 1) & … & r的值。**

int AndSum(int l, int r) {

int ans = r, pos = 0;

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

if ((l & 1) ^ (r & 1)) pos = i;

l >>= 1;

r >>= 1;

}

ans >>= pos;

ans <<= pos;

return ans;

}

**求 l | (l + 1) | … | r的值。**

int OrSum(int l, int r) {

if (l == r) return l;

int pos = 63;

while (!((l ^ r) & 1LL << pos)) --pos;

while (~pos) l |= 1LL << pos--;

return l;

}

**求 l ^ (l + 1) ^ … ^ r的值。**

int PreXorSum(int n) {

int ans = 0;

for (int i = n / 4 \* 4; i <= n; i++) {

ans ^= i;

}

return ans;

}

int XorSum(int l, int r) {

return PreXorSum(r) ^ PreXorSum(l - 1);

}

**LCA**

template <typename T>

struct LCA {

int m;

std::vector<int> d;

std::vector<T> dist;

std::vector<std::vector<pair<int, T>>> g;

std::vector<std::vector<int>> f;

LCA(int n) : d(n + 1), dist(n + 1), g(n + 1), f(n + 1) {

m = \_\_lg(n) + 1;

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

f[i].resize(m + 1);

}

}

void add(int u, int v, T w = 1) {

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

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

}

void build(int root = 1) {

queue<int> q;

q.push(root);

d[root] = 1;

while (!q.empty()) {

int u = q.front(); q.pop();

for (auto [v, w] : g[u]) {

if (d[v]) continue;

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

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

f[v][0] = u;

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

f[v][k] = f[f[v][k - 1]][k - 1];

}

q.push(v);

}

}

}

int lca(int a, int b) {

if (d[a] < d[b]) swap(a, b);

for (int k = m; k >= 0; k--) {

if (d[f[a][k]] >= d[b]) a = f[a][k];

}

if (a == b) return a;

for (int k = m; k >= 0; k--) {

if (f[a][k] != f[b][k]) {

a = f[a][k]; b = f[b][k];

}

}

return f[a][0];

}

T dis(int u, int v) {

return dist[u] + dist[v] - dist[lca(u, v)] \* 2;

}

};

**RMQ\_2D子矩阵为正方形**

struct RMQ\_2D {

int n, m;

std::vector<std::vector<std::vector<int>>> f;

std::vector<std::vector<std::vector<int>>> g;

RMQ\_2D(std::vector<std::vector<int>> a) : n(a.size() - 1), m(a[0].size() - 1) {

f.resize(n + 1);

g.resize(n + 1);

int N = \_\_lg(max(n, m)) + 1;

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

f[i].resize(m + 1);

g[i].resize(m + 1);

for (int j = 0; j <= m; j++) {

f[i][j].resize(N + 1);

g[i][j].resize(N + 1);

}

}

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

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

for (int j = 1; j + (1 << k) - 1 <= m; j++) {

if (k == 0) {

f[i][j][k] = a[i][j];

g[i][j][k] = a[i][j];

} else {

f[i][j][k] = max({f[i][j][k - 1], f[i + (1 << k - 1)][j][k - 1], f[i][j + (1 << k - 1)][k - 1], f[i + (1 << k - 1)][j + (1 << k - 1)][k - 1]});

g[i][j][k] = min({g[i][j][k - 1], g[i + (1 << k - 1)][j][k - 1], g[i][j + (1 << k - 1)][k - 1], g[i + (1 << k - 1)][j + (1 << k - 1)][k - 1]});

}

}

}

}

}

int Max(int x1, int y1, int x2, int y2) {

int k = \_\_lg(x2 - x1 + 1) / \_\_lg(2);

return max({f[x1][y1][k], f[x2 - (1 << k) + 1][y1][k], f[x1][y2 - (1 << k) + 1][k], f[x2 - (1 << k) + 1][y2 - (1 << k) + 1][k]});

};

int Min(int x1, int y1, int x2, int y2) {

int k = \_\_lg(x2 - x1 + 1) / \_\_lg(2);

return min({g[x1][y1][k], g[x2 - (1 << k) + 1][y1][k], g[x1][y2 - (1 << k) + 1][k], g[x2 - (1 << k) + 1][y2 - (1 << k) + 1][k]});

};

};

**RMQ\_2D子矩阵为矩形**

struct RMQ\_2D {

int n, m;

std::vector<std::vector<std::vector<std::vector<int>>>> f;

std::vector<std::vector<std::vector<std::vector<int>>>> g;

RMQ\_2D(std::vector<std::vector<int>> a) : n(a.size() - 1), m(a[0].size() - 1) {

f.resize(n + 1);

g.resize(n + 1);

int N = \_\_lg(n) + 1, M = \_\_lg(m) + 1;

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

f[i].resize(m + 1);

g[i].resize(m + 1);

for (int j = 0; j <= m; j++) {

f[i][j].resize(N + 1);

g[i][j].resize(N + 1);

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

g[i][j][k].resize(M + 1);

f[i][j][k].resize(M + 1);

}

}

}

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

for (int l = 0; l <= M; l++) {

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

for (int j = 1; j + (1 << l) - 1 <= m; j++) {

if (k == 0 && l == 0) {

f[i][j][k][l] = a[i][j];

g[i][j][k][l] = a[i][j];

} else if (k == 0) {

f[i][j][k][l] = max(f[i][j][k][l - 1], f[i][j + (1 << l - 1)][k][l - 1]);

g[i][j][k][l] = min(g[i][j][k][l - 1], g[i][j + (1 << l - 1)][k][l - 1]);

} else if (l == 0) {

f[i][j][k][l] = max(f[i][j][k - 1][l], f[i + (1 << k - 1)][j][k - 1][l]);

g[i][j][k][l] = min(g[i][j][k - 1][l], g[i + (1 << k - 1)][j][k - 1][l]);

} else {

f[i][j][k][l] = max({f[i][j][k - 1][l - 1], f[i + (1 << k - 1)][j][k - 1][l - 1], f[i][j + (1 << l - 1)][k - 1][l - 1], f[i + (1 << k - 1)][j + (1 << l - 1)][k - 1][l - 1]});

g[i][j][k][l] = min({g[i][j][k - 1][l - 1], g[i + (1 << k - 1)][j][k - 1][l - 1], g[i][j + (1 << l - 1)][k - 1][l - 1], g[i + (1 << k - 1)][j + (1 << l - 1)][k - 1][l - 1]});

}

}

}

}

}

}

int Max(int x1, int y1, int x2, int y2) {

int k = \_\_lg(x2 - x1 + 1) / \_\_lg(2);

int l = \_\_lg(y2 - y1 + 1) / \_\_lg(2);

return max({f[x1][y1][k][l], f[x2 - (1 << k) + 1][y1][k][l], f[x1][y2 - (1 << l) + 1][k][l], f[x2 - (1 << k) + 1][y2 - (1 << l) + 1][k][l]});

};

int Min(int x1, int y1, int x2, int y2) {

int k = \_\_lg(x2 - x1 + 1) / \_\_lg(2);

int l = \_\_lg(y2 - y1 + 1) / \_\_lg(2);

return min({g[x1][y1][k][l], g[x2 - (1 << k) + 1][y1][k][l], g[x1][y2 - (1 << l) + 1][k][l], g[x2 - (1 << k) + 1][y2 - (1 << l) + 1][k][l]});

};

};

**树链剖分**

template <typename T>

struct SegmentTree {

struct node {

int l, r;

T s, add;

};

int n;

std::vector<node> tr;

std::vector<T> val;

SegmentTree(int \_n) : n(\_n) {}

void pushup(int u) {

tr[u].s = tr[u << 1].s + tr[u << 1 | 1].s;

}

void pushdown(int u) {

if (tr[u].add == 0) return;

tr[u << 1].s += tr[u].add \* (tr[u << 1].r - tr[u << 1].l + 1);

tr[u << 1 | 1].s += tr[u].add \* (tr[u << 1 | 1].r - tr[u << 1 | 1].l + 1);

tr[u << 1].add += tr[u].add;

tr[u << 1 | 1].add += tr[u].add;

tr[u].add = 0;

}

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

tr[u] = {l, r, 0, 0};

if (l == r) {

tr[u].s = val[l];

return;

}

int mid = l + r >> 1;

build(u << 1, l, mid);

build(u << 1 | 1, mid + 1, r);

pushup(u);

}

void build(std::vector<T> arr) {

val = arr;

tr.resize(n << 2);

build(1, 1, n);

}

void modify(int u, int l, int r, T x) {

if (tr[u].l >= l && tr[u].r <= r) {

tr[u].s += (tr[u].r - tr[u].l + 1) \* x;

tr[u].add += x;

return;

}

int mid = tr[u].l + tr[u].r >> 1;

pushdown(u);

if (l <= mid) modify(u << 1, l, r, x);

if (r > mid) modify(u << 1 | 1, l, r, x);

pushup(u);

}

T query(int u, int l, int r) {

if (tr[u].l >= l && tr[u].r <= r) return tr[u].s;

int mid = tr[u].l + tr[u].r >> 1;

pushdown(u);

T ans = 0;

if(l <= mid) ans += query(u << 1, l, r);

if (r > mid) ans += query(u << 1 | 1, l, r);

return ans;

}

};

template <typename T>

struct TreeChainPartition : SegmentTree<T> {

using SegmentTree<T>::n;

using SegmentTree<T>::modify;

using SegmentTree<T>::query;

std::vector<int> d, f, son, sz;

std::vector<int> dfn, top;

std::vector<T> a, b;

int cnt;

std::vector<std::vector<int>> g;

TreeChainPartition(int \_n) : SegmentTree<T> (\_n) {

g.resize(n + 1);

}

void add(int u, int v) {

g[u].push\_back(v);

g[v].push\_back(u);

}

void dfs1(int u, int fa, int dep) {

d[u] = dep, f[u] = fa, sz[u] = 1;

for (auto v : g[u]) {

if (v == fa) continue;

dfs1(v, u, dep + 1);

sz[u] += sz[v];

if (sz[v] > sz[son[u]]) son[u] = v;

}

}

void dfs2(int u, int t) {

dfn[u] = ++cnt, b[cnt] = a[u], top[u] = t;

if (son[u] == 0) return;

dfs2(son[u], t);

for (auto v : g[u]) {

if (v == f[u] || v == son[u]) continue;

dfs2(v, v);

}

}

void build(std::vector<T> arr) {

a = arr;

b.resize(n + 1);

d.resize(n + 1);

f.resize(n + 1);

son.resize(n + 1);

sz.resize(n + 1);

dfn.resize(n + 1);

top.resize(n + 1);

cnt = 0;

dfs1(1, -1, 1);

dfs2(1, 1);

SegmentTree<T>::build(b);

}

void modify\_path(int u, int v, T k) {

while (top[u] != top[v]) {

if (d[top[u]] < d[top[v]]) swap(u, v);

modify(1, dfn[top[u]], dfn[u], k);

u = f[top[u]];

}

if (d[u] > d[v]) swap(u, v);

modify(1, dfn[u], dfn[v], k);

}

void modify\_tree(int u, T k) {

modify(1, dfn[u], dfn[u] + sz[u] - 1, k);

}

T query\_path(int u, int v) {

T ans = 0;

while (top[u] != top[v]) {

if (d[top[u]] < d[top[v]]) swap(u, v);

ans += query(1, dfn[top[u]], dfn[u]);

u = f[top[u]];

}

if (d[u] > d[v]) swap(u, v);

ans += query(1, dfn[u], dfn[v]);

return ans;

}

T query\_tree(int u) {

return query(1, dfn[u], dfn[u] + sz[u] - 1);

}

};

**单调栈**

std::vector<int> LMin(n + 1);

stack<int> st;

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

while (st.size() && a[st.top()] >= a[i]) st.pop();

LMin[i] = st.size() ? a[st.top()] : -1;

st.push(i);

}

**主席树**

#include <bits/stdc++.h>

using namespace std;

#define int long long

#define endl "\n"

#define debug(var) \

cout << #var << " = " << var << endl

const int N = 2e5 + 10;

int n, m, k;

int len, idx, a[N], b[N], root[N];

struct node {

int l, r, s;

} tr[N \* 21]; // n\*log2(n) 个结点

int find(int x) {

return lower\_bound(b + 1, b + 1 + len, x) - b;

}

int build(int l, int r) {

int u = ++idx;

tr[u].s = 0;

if (l == r) return u;

int mid = l + r >> 1;

tr[u].l = build(l, mid);

tr[u].r = build(mid + 1, r);

return u;

}

int insert(int p, int l, int r, int x) {

int u = ++idx;

tr[u] = tr[p];

if (l == r) return tr[u].s++, u;

int mid = l + r >> 1;

if (x <= mid) tr[u].l = insert(tr[p].l, l, mid, x);

else tr[u].r = insert(tr[p].r, mid+1, r, x);

tr[u].s = tr[tr[u].l].s + tr[tr[u].r].s;

return u;

}

// 大于等于 k 的个数

int query1(int x, int y, int l, int r, int k) {

if (k > b[r]) return 0;

if (k <= b[l]) return tr[x].s - tr[y].s;

int mid = l + r >> 1;

int ans = 0;

if(k <= b[mid]) {

ans += tr[tr[x].r].s - tr[tr[y].r].s;

ans += query1(tr[x].l, tr[y].l, l, mid, k);

} else {

ans += query1(tr[x].r, tr[y].r, mid+1, r, k);

}

return ans;

}

// 小于等于 k 的个数

int query2(int x, int y, int l, int r, int k) {

if(k >= b[r]) return tr[x].s - tr[y].s;

if (k < b[l]) return 0;

int mid = l + r >> 1;

int ans = 0;

if(k <= b[mid]) {

ans += query2(tr[x].l, tr[y].l, l, mid, k);

} else {

ans += tr[tr[x].l].s - tr[tr[y].l].s;

ans += query2(tr[x].r, tr[y].r, mid+1, r, k);

}

return ans;

}

// 大于 k 的个数

int query3(int x, int y, int l, int r, int k) {

if(k >= b[r]) return 0;

if(k < b[l]) return tr[x].s - tr[y].s;

int mid = l + r >> 1;

int ans = 0;

if(k <= b[mid]) {

ans += tr[tr[x].r].s - tr[tr[y].r].s;

ans += query3(tr[x].l, tr[y].l, l, mid, k);

} else {

ans += query3(tr[x].r, tr[y].r, mid+1, r, k);

}

return ans;

}

// 小于 k 的个数

int query4(int x, int y, int l, int r, int k) {

if(k > b[r]) return tr[x].s - tr[y].s;

if(k <= b[l]) return 0;

int mid = l + r >> 1;

int ans = 0;

if(k <= b[mid]) {

ans += query4(tr[x].l, tr[y].l, l, mid, k);

} else {

ans += tr[tr[x].l].s - tr[tr[y].l].s;

ans += query4(tr[x].r, tr[y].r, mid+1, r, k);

}

return ans;

}

// 等于 k 的个数

int query5(int x, int y, int l, int r, int k) {

if(l == r) return k == b[l] ? tr[x].s - tr[y].s : 0;

int mid = l + r >> 1;

if(k <= b[mid]) return query5(tr[x].l, tr[y].l, l, mid, k);

else return query5(tr[x].r, tr[y].r, mid+1, r, k);

}

void solve() {

cin >> n;

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

cin >> a[i]; b[i] = a[i];

}

sort(b + 1, b + 1 + n);

len = unique(b + 1, b + 1 + n) - b - 1;

root[0] = build(1, len);

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

root[i] = insert(root[i-1], 1, len, find(a[i]));

}

/\*------------------------------------\*/

}

signed main() {

ios::sync\_with\_stdio(false);

cin.tie(nullptr);

int T = 1;

// cin >> T;

while (T--) {

solve();

}

return 0 ^ 0;

}

#pragma GCC optimize(1)

#pragma GCC optimize(2)

#pragma GCC optimize(3, "Ofast", "inline")