Team Note of A University

A Team

Compiled on January 9, 2026

Contents

1 Basic Implementation

1.1 Main Template

Usage: Basic CP template with headers and defines.

```
// #pragma GCC optimize ("O3,unroll-loops")
// #pragma GCC target ("avx,avx2,fma") // simd
#include <bits/stdc++.h>
#define fastio cin.tie(0)->sync_with_stdio(0)
#define all(x) (x).begin(),(x).end()
#define rall(x) (x).rbegin(),(x).rend()
#define compress(v) sort(all(v)), v.erase(unique(all(v)), v.end())
#define sz(x) (int)(x).size()
using namespace std;
typedef long long ll;
const ll INF = 1e18;
const int MOD = 998244353;
const int SIZE = 524288;
```

2 Math

2.1 Basic Arithmetic

Usage: Modular inverse, Extended Euclidean.

```
typedef long long ll;
ll modmul(ll a, ll b, ll m) {
    return (__int128)a * b % m;
}
ll modpow(ll b, ll e, ll m) {
  ll ans = 1;
  for (; e; b = modmul(b, b, m), e /= 2)
    if (e & 1) ans = modmul(ans, b, m);
  return ans;
}
ll xgcd(ll a, ll b, ll &x, ll &y) {
    if (!b) return x = 1, y = 0, a;
    ll x1, y1, g = xgcd(b, a % b, x1, y1);
    return x = y1, y = x1 - a / b * y1, g;
}
ll modinv(ll a, ll m) {
    ll x, y;
    ll g = xgcd(a, m, x, y);
    if (g != 1) return -1;
    return (x%m + m) % m;
}
```

2.2 Binomial Coefficient

Time Complexity: $O(1)/O(\sum p^k)$

```
// when M is big prime; Init: O(MAXN), Query: O(1)
typedef long long ll;
ll modmul(ll a, ll b, ll m)
ll modpow(ll b, ll e, ll m)
const int M = 1e9+7, MAXN = 4000000;
ll fac[MAXN+5], finv[MAXN+5];
void init() {
    fac[0] = 1;
    for (int i = 1; i <= MAXN; i++) {
        fac[i] = modmul(fac[i-1], i, M);
    }
    finv[MAXN] = modpow(fac[MAXN], M-2, M);
    for (int i = MAXN-1; i >= 0; i--) {
        finv[i] = modmul(finv[i+1], i+1, M);
    }
}
ll nCk(int n, int k) {
    ll r = modmul(fac[n], finv[n-k], M);
    return modmul(r, finv[k], M);
}

// O(Sum of p^k) per query. (M = product of p^k)
typedef long long ll;
ll modmul(ll a, ll b, ll m)
ll modpow(ll b, ll e, ll m)
ll xgcd(ll a, ll b, ll &x, ll &y)
ll modinv(ll a, ll m)
ll count(ll n, ll p) {
    ll cnt = 0;
    while (n > 0) {
        cnt += n/p; n /= p;
    }
    return cnt;
}
ll calc(ll n, ll p, ll pe, const auto& ft) {
    if (n == 0) return 1;
    ll v = ft[pe], res = modpow(v, n/pe, pe);
    res = modmul(res, ft[n%pe], pe);
    return modmul(res, calc(n/p, p, pe, ft), pe);
}
ll nCk_pe(ll n, ll k, ll p, ll pe, int e) {
    if (k < 0 || k > n) return 0;
    ll pc = count(n, p) - count(k, p) - count(n-k, p);
    if (pc >= e) return 0;
    vector<ll> ft(pe+1); ft[0] = 1;
    for(int i = 1; i <= pe; i++) {
        ft[i] = ft[i-1];
        if (i%p != 0) ft[i] = modmul(ft[i], i, pe);
    }
    ll den = modmul(calc(k, p, pe, ft), calc(n-k, p, pe, ft), pe);
    ll res = modmul(calc(n, p, pe, ft), modinv(den, pe), pe);
    res = modmul(res, modpow(p, pc, pe), pe);
```

```cpp
    return res;
}
ll nCk(ll n, ll k, int m) {
    if (k < 0 || k > n) return 0;
    if (k == 0 || k == n) return 1;
    ll t = m, res = 0;
    auto add = [&](int p, int pe, int e) {
        ll rem = nCk_pe(n, k, p, pe, e);
        ll tm = modmul(rem, m/pe, m);
        tm = modmul(tm, modinv(m/pe, pe), m);
        res = (res + tm) % m;
    };
    for (ll i = 2; i*i <= t; i++) {
        if (t%i == 0) {
            ll p = i, pe = 1; // p^e
            int e = 0;
            while (t%i == 0) {
                pe *= i; t /= i; e++;
            }
            add(p, pe, e);
        }
    }
    if (t > 1) add(t, t, 1);
    return res;
}
```

2.3 Chinese Remainder Theorem

Usage: Solve system of linear congruences.
Time Complexity: $O(\log N)$

```cpp
ll xgcd(ll a, ll b, ll &x, ll &y)
pair<ll,ll> CRT(ll a1, ll m1, ll a2, ll m2) {
    ll x, y, g = xgcd(m1, m2, x, y);
    if ((a2 - a1) % g) return { -1, -1 };
    ll md = m2 / g, k = (a2 - a1) / g % md * (x % md) % md;
    return { a1 + (k < 0 ? k + md : k) * m1, m1 / g * m2 };
}
pair<ll,ll> CRT(const vector<ll>& a, const vector<ll>& m) {
    ll ra = a[0], rm = m[0];
    for (int i = 1; i < (int)m.size(); i++) {
        auto [aa, mm] = CRT(ra, rm, a[i], m[i]);
        if (mm == -1) return { -1, -1 };
        ra = aa; rm = mm;
    }
    return { ra, rm };
}
```

2.4 FFT & NTT

Usage: Fast Fourier/Number Theoretic Transform for convolutions.
Time Complexity: $O(N \log N)$

```cpp
typedef long long ll;
template<int M> struct MINT {
    int v;
    MINT(ll _v = 0) { v = _v % M; if (v < 0) v += M; }
    MINT operator+(const MINT& o) const { return MINT(v + o.v); }
    MINT operator-(const MINT& o) const { return MINT(v - o.v); }
    MINT operator*(const MINT& o) const { return MINT((ll)v * o.v); }
    MINT& operator*=(const MINT& o) { return *this = *this * o; }
    friend MINT pw(MINT a, ll b) {
        MINT r = 1; for (; b; b >>= 1, a *= a) if (b & 1) r *= a;
        return r;
    }
    friend MINT inv(MINT a) { return pw(a, M - 2); }
};
namespace fft {
    using cpx = complex<double>;
    void rev_bit(int n, vector<auto>& a) {
        for (int i = 1, j = 0; i < n; i++) {
            int bit = n >> 1; for (; j & bit; bit >>= 1) j ^= bit; j ^= bit;
            if (i < j) swap(a[i], a[j]);
        }
    }
    void FFT(vector<cpx>& a, bool inv_f) {
        int n = a.size(); rev_bit(n, a);
        for (int len = 2; len <= n; len <<= 1) {
            double ang = 2 * acos(-1) / len * (inv_f ? -1 : 1);
            cpx wlen(cos(ang), sin(ang));
            for (int i = 0; i < n; i += len) {
                cpx w(1);
                for (int j = 0; j < len / 2; j++) {
                    cpx u = a[i + j], v = a[i + j + len / 2] * w;
                    a[i + j] = u + v; a[i + j + len / 2] = u - v; w *= wlen;
                }
            }
        }
        if (inv_f) for (auto& x : a) x /= n;
    }
    vector<ll> multiply(const vector<ll>& a, const vector<ll>& b) {
        int n = 1; while (n < a.size() + b.size()) n <<= 1;
        vector<cpx> fa(n), fb(n);
        for(int i=0; i<a.size(); i++) fa[i] = cpx(a[i], 0);
        for(int i=0; i<b.size(); i++) fb[i] = cpx(b[i], 0);
        FFT(fa, 0); FFT(fb, 0);
        for(int i=0; i<n; i++) fa[i] *= fb[i];
        FFT(fa, 1);
        vector<ll> res(n);
        for(int i=0; i<n; i++) res[i] = llround(fa[i].real());
        return res;
    }
    vector<ll> multiply_mod(const vector<ll>& a, const vector<ll>& b, ll mod) {
        int n = 1; while (n < a.size() + b.size()) n <<= 1;
        vector<cpx> v1(n), v2(n), r1(n), r2(n);
        for (int i = 0; i < a.size(); i++) v1[i] = cpx(a[i] >> 15, a[i] & 32767);
        for (int i = 0; i < b.size(); i++) v2[i] = cpx(b[i] >> 15, b[i] & 32767);
        FFT(v1, 0); FFT(v2, 0);
        for (int i = 0; i < n; i++) {
            int j = i ? n - i : i;
            cpx a1 = (v1[i] + conj(v1[j])) * cpx(0.5, 0), a2 = (v1[i] - conj(v1[j]))
            * cpx(0, -0.5);
            cpx b1 = (v2[i] + conj(v2[j])) * cpx(0.5, 0), b2 = (v2[i] - conj(v2[j]))
            * cpx(0, -0.5);
            r1[i] = a1 * b1 + a1 * b2 * cpx(0, 1); r2[i] = a2 * b1 + a2 * b2 * cpx(0,
            1);
        }
        FFT(r1, 1); FFT(r2, 1);
        vector<ll> res(n);
        for (int i = 0; i < n; i++) {
            ll av = (ll)round(r1[i].real()) % mod, cv = (ll)round(r2[i].imag()) %
            mod;
            ll bv = ((ll)round(r1[i].imag()) + (ll)round(r2[i].real())) % mod;
            res[i] = (av << 30) + (bv << 15) + cv; res[i] = (res[i] % mod + mod) %
            mod;
        }
        return res;
    }
    template<int W, int M> void NTT(vector<MINT<M>>& a, bool inv_f) {
        int n = a.size(); rev_bit(n, a);
        for (int len = 2; len <= n; len <<= 1) {
            MINT<M> wlen = pw(MINT<M>(W), (M - 1) / len);
            if (inv_f) wlen = inv(wlen);
            for (int i = 0; i < n; i += len) {
                MINT<M> w = 1;
                for (int j = 0; j < len / 2; j++) {
                    MINT<M> u = a[i + j], v = a[i + j + len / 2] * w;
                    a[i + j] = u + v; a[i + j + len / 2] = u - v; w *= wlen;
                }
            }
        }
        if (inv_f) { MINT<M> rn = inv(MINT<M>(n)); for (auto& x : a) x *= rn; }
    }
}
template<int W, int M> struct Poly {
    using T = MINT<M>; vector<T> a;
    Poly(const vector<T>& _a = {}) : a(_a) { norm(); }
    void norm() { while (a.size() && a.back().v == 0) a.pop_back(); }
    int deg() const { return (int)a.size() - 1; }
    T operator[](int i) const { return i < a.size() ? a[i] : T(0); }
    Poly operator*(const Poly& o) const {
        if (a.empty() || o.a.empty()) return {};
        int n = 1, sz = a.size() + o.a.size() - 1;
        while (n < sz) n <<= 1;
        vector<T> fa(n), fb(n); copy(all(a), fa.begin()); copy(all(o.a), fb.begin());
        fft::NTT<W, M>(fa, 0); fft::NTT<W, M>(fb, 0);
        for (int i = 0; i < n; i++) fa[i] *= fb[i];
        fft::NTT<W, M>(fa, 1); return fa;
    }
    Poly inv(int n) const {
        Poly r({ ::inv(a[0]) });
        for (int i = 1; i < n; i <<= 1) {
            Poly tmp(vector<T>(a.begin(), a.begin() + min((int)a.size(), i * 2)));
            r = (r * (Poly({T(2)}) - r * tmp)); r.a.resize(i * 2);
        }
        r.a.resize(n); return r;
    }
    Poly operator/(Poly o) const {
        if (deg() < o.deg()) return {};
        int n = deg() - o.deg() + 1;
        Poly ra = a, rb = o.a; reverse(all(ra.a)); reverse(all(rb.a));
        Poly q = (ra * rb.inv(n)); q.a.resize(n); reverse(all(q.a)); return q;
    }
    Poly operator%(Poly o) const {
        if (deg() < o.deg()) return *this;
        Poly r = *this - (*this / o) * o; r.norm(); return r;
    }
    Poly operator-(const Poly& o) const {
        vector<T> res(max(a.size(), o.a.size()));
        for (int i = 0; i < res.size(); i++) res[i] = (*this)[i] - o[i];
        return res;
    }
};
using mint = MINT<998244353>;
using poly = Poly<3, 998244353>;
mint Kitamasa(poly c, poly a, ll n) {
    if (n <= a.deg()) return a[n];
    poly f; for (int i = 0; i <= c.deg(); i++) f.a.push_back(mint(0) - c[c.deg() -
    i]);
    f.a.push_back(1);
    poly res({1}), x({0, 1});
    for (; n; n >>= 1, x = (x * x) % f) if (n & 1) res = (res * x) % f;
    mint ans = 0; for (int i = 0; i <= a.deg(); i++) ans = ans + a[i] * res[i];
    return ans;
}
int main() {
    vector<ll> A = {1, 2, 1}; // 1+2*x+x^2
    vector<ll> B = {1, 1};    // 1+x
    vector<ll> C = fft::multiply(A, B); // {1, 3, 3, 1}
    vector<ll> D = fft::multiply_mod(A, B, 1000000007);
    poly p1({1, 2, 1}), p2({1, 1}); // NTT base
    p1 * p2; p1 / p2; p1 % p2; // polynomial operation
    // A_n = 1*A_{n-1} + 1*A_{n-2}
    poly coeffs({1, 1}); // {c0, c1} 순서 (A_{n-2}, A_{n-1} 계수)
    poly initial({0, 1}); // {A0, A1} 초기값
    cout << Kitamasa(coeffs, initial, 1000000000).v;
}
```

2.5 Linear Sieve

Usage: Find primes and multiplicative functions in linear time.
Time Complexity: $O(N)$

```
vector<int> Linear_sieve(int N) {
    vector<int> sieve(N+1, 1), prime(N+1);
    int pcnt = 0;
    for (int i = 2; i <= N; i++) {
        if (sieve[i]) prime[pcnt++] = i;
        for (int j = 0; i*prime[j] <= N; j++) {
            sieve[i*prime[j]] = 0;
            if (i%prime[j] == 0) break;
        }
    }
    prime.resize(pcnt);
    return prime;
}
```

2.6 Miller-Rabin & Pollard-Rho

Usage: Primality test and integer factorization.
Time Complexity: $O(\log^3 N)/O(N^{1/4})$

```
ll modmul(ll a, ll b, ll m)
ll modpow(ll b, ll e, ll m)
bool isPrime(ll n) {
    if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
    ll A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
        s = __builtin_ctzll(n-1), d = n >> s;
    for (ll a : A) {   // ^ count trailing zeroes
        ll p = modpow(a%n, d, n), i = s;
        while (p != 1 && p != n - 1 && a % n && i--)
            p = modmul(p, p, n);
        if (p != n-1 && i != s) return 0;
    }
    return 1;
}
ll pollard(ll n) {
    auto f = [n](ll x) { return modmul(x, x, n) + 3; };
    ll x = 0, y = 0, t = 30, prd = 2, i = 1, q;
    while (t++ % 40 || __gcd(prd, n) == 1) {
        if (x == y) x = ++i, y = f(x);
        if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd = q;
        x = f(x), y = f(f(y));
    }
    return __gcd(prd, n);
}
vector<ll> factor(ll n) {
    if (n == 1) return {};
    if (isPrime(n)) return {n};
    ll x = pollard(n);
    auto l = factor(x), r = factor(n / x);
    l.insert(l.end(), r.begin(), r.end());
    return l;
}
vector<ll> res = factor(N); // factor of N
```

3 Data Structure

3.1 Erasable Priority Queue

Usage: Priority queue supporting arbitrary element deletion.

```
template <typename T = int, typename Compare = std::less<T>>
struct EraseablePQ {
    priority_queue<T,vector<T>,Compare> q, del;
    void flush() {
        while (!del.empty() && !q.empty() && q.top() == del.top()) {
            q.pop(); del.pop();
        }
    }
    void push(const T& x) { q.push(x); flush(); }
    void erase(const T& x) { del.push(x); flush(); }
    void pop() { flush(); if (!q.empty()) q.pop(); flush(); }
    const T& top() { flush(); return q.top(); }
    int size() const { return int(q.size() - del.size()); }
    bool empty() { flush(); return q.empty(); }
};
```

3.2 Non-Recursive Segment Tree

Usage: Segment tree for performance.
Time Complexity: $O(\log N)$/ query

```
template<typename Node>
struct SegTree {
    int n, lg, size;
    Node e; // 항등원
    vector<Node> tree;
    function<Node(Node, Node)> func;
    int log2(int n) {
        int res = 0;
        while (n > (1 << res)) res++;
        return res;
    }
    SegTree(int n, const Node& e, auto func) : n(n), lg(log2(n)), size(1<<lg), e(e),
    tree(size<<1, e), func(func) {}
    SegTree(const vector<Node>& v, const Node& e, auto func) : n(sz(v)), lg(log2(n)),
    size(1<<lg), e(e), tree(size<<1, e), func(func) {
        for (int i = 0; i < n; i++) {
            tree[i+size] = v[i];
        }
        for (int i = size-1; i > 0; i--) {
            tree[i] = func(tree[i<<1], tree[i<<1 | 1]);
        }
    }
    void add(int i, const Node& val) {
        tree[--i |= size] += val;
        while (i >>= 1) {
            tree[i] = func(tree[i<<1], tree[i<<1 | 1]);
        }
    }
    void update(int i, const Node& val) {
        tree[--i |= size] = val;
        while (i >>= 1) {
            tree[i] = func(tree[i<<1], tree[i<<1 | 1]);
        }
    }
    Node query(int i) {
        return tree[--i | size];
    }
    Node query(int l, int r) {
        Node L = e, R = e;
        for (--l |= size, --r |= size; l <= r; l >>= 1, r >>= 1) {
            if (l & 1) L = func(L, tree[l++]);
            if (~r & 1) R = func(tree[r--], R);
        }
        return func(L, R);
    }
    int find_kth(Node k) {
        int node = 1, st = 1, en = size;
        while (st != en) {
            int mid = (st + en) / 2;
            node <<= 1;
            if (tree[node] >= k) en = mid;
            else {
                k -= tree[node];
                node |= 1; st = mid+1;
            }
        }
        return st;
    }
};
int main() {
    // 1. Range Sum Query (RSQ)
    vector<int> v = {1, 2, 3, 4, 5};
    SegTree<int> rsq(v, 0, [](int a, int b) { return a+b; });
    rsq.update(3, 10);
    int sum = rsq.query(2, 4);
    // 2. Range Minimum Query (RMQ)
    const int INF = 1e9;
    SegTree<int> rmq(N, INF, [](int a, int b) { return min(a, b); });
    // 3. Binary Search on Tree (Order Statistic)
    // - Requirement: The tree must represent frequency or counts.
    // - Find the smallest index i such that prefix_sum(1...i) >= k
    int idx = rsq.find_kth(7);
}
```

3.3 Merge Sort Tree

Usage: Count/rank of elements in range $[L, R]$.
Time Complexity: $O(\log^2 N)$/ query

```
template <typename T>
struct MergeSortTree {
    int sz;
    vector<vector<T>> tree; // Space: O(N log N)
    MergeSortTree(int n) {
        sz = 1;
        while (sz < n) sz <<= 1;
        tree.resize(sz * 2);
    }
    void add(int x, T v) { tree[x + sz].push_back(v); }
    void build() { // Build: O(N log N)
        for (int i = sz - 1; i > 0; i--) {
            tree[i].resize(tree[i * 2].size() + tree[i * 2 + 1].size());
            merge(tree[i * 2].begin(), tree[i * 2].end(),
                  tree[i * 2 + 1].begin(), tree[i * 2 + 1].end(),
                  tree[i].begin());
        }
    }
    int query(int l, int r, T k) { // Query: O(log^2 N)
        int res = 0;
        for (l += sz, r += sz; l <= r; l >>= 1, r >>= 1) {
            if (l & 1) {
                res += tree[l].end() - upper_bound(tree[l].begin(), tree[l].end(),
                k);
                l++;
            }
            if (!(r & 1)) {
                res += tree[r].end() - upper_bound(tree[r].begin(), tree[r].end(),
                k);
                r--;
            }
            /*
             - Count < k: lower_bound(all(v)) - v.begin()
             - Count <= k: upper_bound(all(v)) - v.begin()
             - Count >= k: v.end() - lower_bound(all(v))
             - Count > k: v.end() - upper_bound(all(v))
            */
        }
        return res;
    }
};
```

3.4 Persistent Segment Tree

Usage: Accessing previous versions and range k-th element.

Time Complexity: $O(\log N)$/ query

```
struct PSTNode{
    PSTNode *l, *r; int v;
    PSTNode(){ l = r = nullptr; v = 0; }
};
PSTNode *root[101010];
PST(){ memset(root, 0, sizeof root); } // constructor
void init(PSTNode *node, int s, int e){
    if(s == e) return;
    int m = s + e >> 1;
    node->l = new PSTNode; node->r = new PSTNode;
    init(node->l, s, m); init(node->r, m+1, e);
}
void update(PSTNode *prv, PSTNode *now, int s, int e, int x){
    if(s == e){ now->v = prv ? prv->v + 1 : 1; return; }
    int m = s + e >> 1;
    if(x <= m){
        now->l = new PSTNode; now->r = prv->r;
        update(prv->l, now->l, s, m, x);
    }
    else{
        now->r = new PSTNode; now->l = prv->l;
        update(prv->r, now->r, m+1, e, x);
    }
    int t1 = now->l ? now->l->v : 0;
    int t2 = now->r ? now->r->v : 0;
    now->v = t1 + t2;
}
int kth(PSTNode *prv, PSTNode *now, int s, int e, int k){
    if(s == e) return s;
    int m = s + e >> 1, diff = now->l->v - prv->l->v;
    if(k <= diff) return kth(prv->l, now->l, s, m, k);
    else return kth(prv->r, now->r, m+1, e, k-diff);
}
```

3.5 Sweepline Mo's

Usage: Optimized Mo's for $O(1)$ update and $O(1)$ query via offline sweepline.
Time Complexity: $O(N\sqrt{Q})$

```
typedef long long ll;
const int MAXN = 200005;
const int BSIZ = 450;

struct SqrtDecomp {
    ll lz_v[BSIZ+5], lz_c[BSIZ+5], v_arr[MAXN], c_arr[MAXN];
    ll total_v = 0, total_c = 0;
    void clear() { memset(this, 0, sizeof(*this)); }
    void update(int idx, ll v) { // O(sqrt N)
        total_v += v; total_c++;
        int b = idx / BSIZ;
        for (int i = idx; i < (b + 1) * BSIZ && i < MAXN; i++) {
            v_arr[i] += v; c_arr[i]++;
        }
        for (int i = b + 1; i <= BSIZ; i++) {
            lz_v[i] += v; lz_c[i]++;
        }
    }
    ll query(int idx, ll v) { // O(1)
        if (idx < 0) return (total_c * v - total_v); // 필요 시 수정
        ll cur_v = lz_v[idx / BSIZ] + v_arr[idx];
        ll cur_c = lz_c[idx / BSIZ] + c_arr[idx];
        return (cur_c * v - cur_v) + ((total_v - cur_v) - (total_c - cur_c) * v);
    }
} sd;

struct MoSweep {
    struct Query {
        int l, r, id; ll ans;
        bool operator<(const Query& o) const {
            if (l / BSIZ != o.l / BSIZ) return l < o.l;
            return (l / BSIZ) & 1 ? r < o.r : r > o.r;
        }
    };
    struct Delta { int q_idx, l, r; bool is_sub; };
    int n, q;
    ll A[MAXN], pref[MAXN], result[MAXN], rnk[MAXN];
    vector<Query> queries, sweep[MAXN];
    void init(int _n) {
        n = _n; queries.clear();
        for(int i = 0; i<=n; i++) sweep[i].clear();
    }
    void add_query(int l, int r, int id) { queries.push_back({l, r, id, 0}); }
    void build() {
        sort(queries.begin(), queries.end());
        sd.clear();
        for (int i = 1; i <= n; i++) {
            pref[i] = sd.query(rnk[i], A[i]);
            sd.update(rnk[i], A[i]);
        }
        int s = 1, e = 0;
        for (int i = 0; i < (int)queries.size(); i++) {
            int nl = queries[i].l, nr = queries[i].r;
            if (e < nr) sweep[s - 1].push_back({i, e + 1, nr, true}), e = nr;
            if (s > nl) sweep[e].push_back({i, nl, s - 1, false}), s = nl;
            if (e > nr) sweep[s - 1].push_back({i, nr + 1, e, false}), e = nr;
            if (s < nl) sweep[e].push_back({i, s, nl - 1, true}), s = nl;
        }
    }
    void solve() {
        sd.clear();
        for (int i = 1; i <= n; i++) {
            sd.update(rnk[i], A[i]);
            for (auto& d : sweep[i]) {
                ll tmp = 0;
                for (int k = d.l; k <= d.r; k++) tmp += sd.query(rnk[k], A[k]);
                queries[d.q_idx].ans += (d.is_sub ? -tmp : tmp);
            }
        }
        int s = 1, e = 0;
        for (int i = 0; i < (int)queries.size(); i++) {
            while (e < queries[i].r) queries[i].ans += pref[++e];
            while (s > queries[i].l) queries[i].ans -= pref[--s];
            while (e > queries[i].r) queries[i].ans -= pref[e--];
            while (s < queries[i].l) queries[i].ans += pref[s++];
            if (i > 0) queries[i].ans += queries[i - 1].ans;
            result[queries[i].id] = queries[i].ans;
        }
    }
} engine;

int main() {
    int n, q; cin >> n >> q;
    engine.init(n);
    vector<pair<ll,int>> v(n);
    for (int i = 1; i <= n; i++) {
        cin >> engine.A[i];
        v[i - 1] = {engine.A[i], i};
    }
    sort(v.begin(), v.end());
    for (int i = 0; i < n; i++) engine.rnk[v[i].second] = i;
    for (int i = 0; i < q; i++) {
        int l, r; cin >> l >> r;
        engine.add_query(l, r, i);
    }
    engine.build();
    engine.solve();
    for (int i = 0; i < q; i++) cout << engine.result[i] << "\n";
    return 0;
}
```

4 Graph

4.1 Bellman Ford

Usage: SSSP with negative weights/cycles.
Time Complexity: $O(VE)$

```
int N, M; ll D[555]; // 주의: 웬만하면 long long으로 잡는 게 좋음
vector<tuple<int,int,ll>> E; // {from, to, weight}
void AddEdge(int s, int e, int w){
    E.emplace_back(s, e, w);
}
// s에서 도달 가능한 음수 사이클 있으면 false 반환
bool Run(int s){
    memset(D, 0x3f, sizeof D);
    ll INF = D[0];
    D[s] = 0;
    for(int iter=1; iter<=N; iter++){
        bool changed = false;
        for(auto [u, v, w] : E){
            if(D[u] == INF) continue;
            if(D[v] > D[u] + w) D[v] = D[u] + w, changed = true;
        }
        if(iter == N && changed) return false;
    }
    return true;
}
```

4.2 LCA

Usage: Lowest Common Ancestor using binary lifting.
Time Complexity: $O(\log N)$

```
int N, Q, D[101010], P[22][101010];
vector<int> G[101010];
void Connect(int u, int v){
    G[u].push_back(v); G[v].push_back(u);
}
void DFS(int v, int b=-1){
    for(auto i : G[v]) if(i != b) D[i] = D[v] + 1, P[0][i] = v, DFS(i, v);
}
int LCA(int u, int v){
    if(D[u] < D[v]) swap(u, v);
    int diff = D[u] - D[v];
    for(int i=0; diff; i++, diff>>=1) if(diff & 1) u = P[i][u];
    if(u == v) return u;
    for(int i=21; i>=0; i--) if(P[i][u] != P[i][v]) u = P[i][u], v = P[i][v];
    return P[0][u];
}
////
// 1. Connect로 간선 추가
// 2. DFS(1) 호출
// 3. 아래 코드 실행
for(int i=1; i<22; i++) for(int j=1; j<=N; j++) P[i][j] = P[i-1][P[i-1][j]];
// 4. LCA(u, v)로 최소 공통 조상 구할 수 있음
```

4.3 HLD

Usage: Heavy-Light Decomposition for path queries on trees.
Time Complexity: $O(\log^2 N)$

```
struct HLD{
    vector<int> dep, par, sz, in, out, top;
    int idx;
    vector<vector<int>> adj, graph;
    int n;
    HLD (int n_){
        n = n_;
        idx = 0;
        dep.resize(n+1);
        par.resize(n+1);
        sz.resize(n+1);
        in.resize(n+1);
        out.resize(n+1);
        top.resize(n+1);
        adj.resize(n+1);
        graph.resize(n+1);
    }
    void addEdge(int u, int v){
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    void dfs(int v=1, int pre=-1){
        for (int u : adj[v]){
            if (u == pre) continue;

            graph[v].push_back(u);
            dfs(u, v);
        }
    }
    void dfs1(int v=1){
        sz[v] = 1;
        for (int &u : graph[v]){
            dep[u] = dep[v] + 1;
            par[u] = v;
            dfs1(u);
            sz[v] += sz[u];

            if (sz[u] > sz[graph[v][0]]) swap(u, graph[v][0]);
        }
    }
    void dfs2(int v=1){
        in[v] = ++idx;
        for (int u : graph[v]){
            top[u] = (u == graph[v][0]) ? top[v] : u;
            dfs2(u);
        }
        out[v] = idx;
    }
    void calculate(){
        dfs(); dfs1(); dfs2();
    }
    array<vector<array<int, 2>>, 2> getPath(int u, int v){
        vector<array<int, 2>> v1, v2;
        while (top[u] != top[v]){
            if (dep[top[u]] > dep[top[v]]){
                ll xx = top[u];
                v1.push_back({in[xx], in[u]});
                u = par[xx];
            }else{
                ll xx = top[v];
                v2.push_back({in[xx], in[v]});
                v = par[xx];
            }
        }
        if (dep[u] < dep[v]){
            v2.push_back({in[u], in[v]});
        }else{
            v1.push_back({in[v], in[u]});
        }
        return {v1, v2};
        // auto pp = hld.getPath(u, v);
        // Node res1 = id;
        // Node res2 = id;
        // for (auto p2 : pp[0]){
        //     res1 = seg.merge(seg.query(p2[0], p2[1]+1), res1);
        // }
        // for (auto p2 : pp[1]){
        //     res2 = seg.merge(seg.query(p2[0], p2[1]+1), res2);
        // }
        // swap(res1.lsum, res1.rsum);
        // auto res = seg.merge(res1, res2);
    }
};
```

4.4 Centroid Decomposition

Usage: Divide and conquer on trees for path/distance problems.
Time Complexity: $O(N \log N)$ build

```
struct CentroidTree {
    int N;
    vector<vector<int>> adj;      // 원본 트리
    vector<vector<int>> c_adj;    // 센트로이드 트리
    vector<int> sz, par, vis;
    CentroidTree(int n) : N(n), adj(n + 1), c_adj(n + 1), sz(n + 1), par(n + 1),
    vis(n + 1, 0) {}
    void add_edge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    int get_sz(int curr, int prev) {
        sz[curr] = 1;
        for (int next : adj[curr])
            if (next != prev && !vis[next]) sz[curr] += get_sz(next, curr);
        return sz[curr];
    }
    int get_centroid(int curr, int prev, int total_sz) {
        for (int next : adj[curr])
            if (next != prev && !vis[next] && sz[next] > total_sz / 2)
```

```
                return get_centroid(next, curr, total_sz);
        return curr;
    }
    // static_solve: 현재 센트로이드를 포함하는 모든 경로를 계산하는 로직
    void static_solve(int u) {
        /* */
    }
    int build(int curr, int p = -1) {
        int centroid = get_centroid(curr, -1, get_sz(curr, -1));
        static_solve(centroid); // 정적 분할 정복 문제일 때 사용
        vis[centroid] = 1;
        par[centroid] = p;
        for (int next : adj[centroid]) {
            if (!vis[next]) {
                int child = build(next, centroid);
                c_adj[centroid].push_back(child); // 센트로이드 계층 연결
            }
        }
        return centroid;
    }
};
```

4.5 Bipartite Matching

Usage: Maximum matching in bipartite graphs.
Time Complexity: $O(E\sqrt{V})$

```
struct BiMatch { // Hopcroft-Karp
    vector<vector<int>> graph, grev;
    vector<int> mA, mB, dist, work, vis;
    int ns, ms;
    BiMatch(int n, int m) : ns(n), ms(m) {
        graph.resize(n+1);
        grev.resize(m+1);
        mA.resize(n+1);
        mB.resize(m+1);
        dist.resize(n+1);
        work.resize(n+1);
        vis.resize(max(n,m)+1);
    }
    void add(int a, int b) { graph[a].push_back(b); grev[b].push_back(a); }
    void bfs() {
        fill(all(dist), -1);
        queue<int> q;
        for (int i = 1; i <= ns; i++) {
            if (!mA[i]) {
                dist[i] = 0;
                q.push(i);
            }
        }
        while (!q.empty()) {
            int i = q.front(); q.pop();
            for (auto j : graph[i]) {
                int k = mB[j];
                if (k && dist[k] == -1) {
                    dist[k] = dist[i] + 1;
                    q.push(k);
                }
            }
        }
    }
    bool dfs(int cur) {
        for (int& i = work[cur]; i < sz(graph[cur]); i++) {
            int nb = graph[cur][i];
            int ori = mB[nb];
            if (!ori || dist[ori] == dist[cur] + 1 && dfs(ori)) {
                mA[cur] = nb;
                mB[nb] = cur;
                return true;
            }
        }
        return false;
    }
    int match() {
        int ans = 0;
        while (1) {
            fill(all(work), 0);
            bfs();
            int cnt = 0;
            for (int i = 1; i <= ns; i++) {
                if (!mA[i] && dfs(i)) cnt++;
            }
            if (!cnt) break;
            ans += cnt;
        }
        return ans;
    }
    pair<vector<int>,vector<int>> vertex() { // find minimum vertex cover
        vector<bool> visA(ns+1), visB(ms+1);
        queue<int> q;
        for (int i = 1; i <= ns; i++) {
            if (!mA[i]) {
                q.push(i);
                visA[i] = 1;
            }
        }
        while (!q.empty()) {
            int a = q.front(); q.pop();
            for (auto b : graph[a]) {
                if (!visB[b]) {
                    visB[b] = 1;
                    int na = mB[b];
                    if (na && !visA[na]) {
                        visA[na] = 1;
                        q.push(na);
                    }
                }
            }
        }
        vector<int> retA, retB;
        for (int i = 1; i <= ns; i++) {
            if (visA[i]) retA.push_back(i);
        }
        for (int i = 1; i <= ms; i++) {
            if (!visB[i]) retB.push_back(i);
        }
        return { retA, retB };
    }
    bool dfs1(int a, int b) {
        vis[a] = 1;
        for (auto i : graph[a]) {
            if (i == b) continue;
            int ori = mB[i];
            if (ori == 0 || (!vis[ori] && dfs1(ori, b))) {
                return true;
            }
        }
        return false;
    }
    bool dfs2(int b, int a) {
        vis[b] = 1;
        for (auto i : grev[b]) {
            if (i == a) continue;
            int ori = mA[i];
            if (ori == 0 || (!vis[ori] && dfs2(ori, a))) {
                return true;
            }
        }
        return false;
    }
    bool chk1(int a) { // find max matching except a
        fill(all(vis), 0);
        int b = mA[a];
        return (b == 0) || dfs2(b, a);
    }
    bool chk2(int b) { // find max matching except b
        fill(all(vis), 0);
        int a = mB[b];
        return (a == 0) || dfs1(a, b);
    }
};

/*
struct BiMatch {
    vector<vector<int>> graph;
    vector<int> mA, mB, vis;
    int ns, ms;
    BiMatch(int n, int m) : ns(n), ms(m) {
        graph.resize(n+1);
        mA.resize(n+1);
        mB.resize(m+1);
        vis.resize(n+1);
    }
    void add(int a, int b) { graph[a].push_back(b); }
    bool dfs(int cur) {
        vis[cur] = 1;
        for (auto i : graph[cur]) {
            int ori = mB[i];
            if (ori == 0 || (!vis[ori] && dfs(ori))) {
                mA[cur] = i;
                mB[i] = cur;
                return true;
            }
        }
        return false;
    }
    int match() {
        int res = 0;
        for (int i = 1; i <= ns; i++) {
            if (mA[i]) continue;
            fill(all(vis), 0);
            if (dfs(i)) res++;
        }
        return res;
    }
};
*/
```

4.6 Dinic

Usage: Efficient maximum flow algorithm.
Time Complexity: $O(V^2E)$

```
typedef long long ll;
const ll INF = 1e18;
struct Dinic {
    struct Edge {
        int to; ll cap; int rev;
    };
    vector<vector<Edge>> graph;
    vector<int> level, work;
    int n;
    Dinic(int n) : n(n) {
        graph.resize(n+1);
        level.resize(n+1);
        work.resize(n+1);
    }
    void add(int u, int v, ll cap) {
        graph[u].push_back({ v, cap, sz(graph[v])});
        graph[v].push_back({ u, 0, sz(graph[u])-1});
    }
    bool bfs(int s, int t) {
        fill(all(level), -1); level[s] = 0;
        queue<int> q; q.push(s);
        while (!q.empty()) {
            int cur = q.front(); q.pop();
            for (auto [nxt, cap, rev] : graph[cur]) {
                if (cap > 0 && level[nxt] == -1) {
                    level[nxt] = level[cur]+1;
                    q.push(nxt);
                }
            }
        }
        return (level[t] != -1);
    }
    ll dfs(int cur, int t, ll flow) {
        if (cur == t) return flow;
        for (int& i = work[cur]; i < sz(graph[cur]); i++) {
            auto& [nxt, cap, rev] = graph[cur][i];
            if (cap > 0 && level[nxt] == level[cur]+1) {
                ll push = dfs(nxt, t, min(flow, cap));
                if (push > 0) {
                    cap -= push;
                    graph[nxt][rev].cap += push;
                    return push;
                }
            }
        }
        return 0;
    }
    ll flow(int s, int t) {
        ll ans = 0;
        while (bfs(s, t)) {
            fill(all(work), 0);
            while (auto flow = dfs(s, t, INF)) {
                ans += flow;
            }
        }
        return ans;
    }
    vector<bool> mincut(int s) {
        vector<bool> vis(n+1); vis[s] = true;
        queue<int> q; q.push(s);
        while (!q.empty()) {
            int cur = q.front(); q.pop();
            for (auto [nxt, cap, rev] : graph[cur]) {
                if (cap > 0 && !vis[nxt]) {
                    vis[nxt] = true;
                    q.push(nxt);
                }
            }
        }
        return vis;
    }
};
```

4.7 MCMF

Usage: Minimum Cost Maximum Flow using SPFA.
Time Complexity: $O(F \cdot E \log V)$

```
typedef long long ll;
const ll INF = 1e18;
struct MCMF {
    struct Edge {
        int to; ll cap, cost; int rev;
    };
    vector<vector<Edge>> graph;
    vector<ll> dist;
    vector<int> parent, edge;
    vector<bool> vis;
    int n;
    MCMF(int n) : n(n) {
        graph.resize(n+1);
        dist.resize(n+1);
        parent.resize(n+1);
        edge.resize(n+1);
        vis.resize(n+1);
    }
    void add(int u, int v, ll cap, ll cost) {
        graph[u].push_back({ v, cap, cost, sz(graph[v]) });
        graph[v].push_back({ u, 0, -cost, sz(graph[u])-1 });
    }
    bool spfa(int s, int t) {
        fill(all(dist), INF);
        fill(all(parent), -1);
        fill(all(vis), false);
        queue<int> q; q.push(s);
        dist[s] = 0; vis[s] = true;
        while (!q.empty()) {
            int cur = q.front(); q.pop();
            vis[cur] = false;
            for (int i = 0; i < sz(graph[cur]); i++) {
                auto& [nxt, cap, cost, rev] = graph[cur][i];
                if (cap > 0 && dist[nxt] > dist[cur] + cost) {
                    dist[nxt] = dist[cur] + cost;
                    parent[nxt] = cur;
                    edge[nxt] = i;
                    if (!vis[nxt]) {
                        vis[nxt] = true;
                        q.push(nxt);
                    }
                }
            }
        }
        return dist[t] != INF;
    }
    pair<int,ll> flow(int s, int t) {
        int res = 0; ll cost = 0;
        while (spfa(s, t)) {
            ll fl = INF;
            for (int v = t; v != s; v = parent[v]) {
                int u = parent[v], idx = edge[v];
                fl = min(fl, graph[u][idx].cap);
            }
            for (int v = t; v != s; v = parent[v]) {
                int u = parent[v], idx = edge[v], ridx = graph[u][idx].rev;
                graph[u][idx].cap -= fl;
                graph[v][ridx].cap += fl;
                cost += (ll)fl * graph[u][idx].cost;
            }
            res += fl;
        }
        return { res, cost };
    }
};
```

4.8 Circulation

Usage: Flow with lower and upper bounds.

```
typedef long long ll;
const ll INF = 1e18;
struct Dinic {}; // or MCMF
struct Circulation {
    int n, S, T;
    vector<ll> demand, low;
    vector<pair<int,int>> edge;
    Dinic dn;
    Circulation(int n) : n(n), S(n+1), T(n+2), demand(n+3, 0), dn(n+2) {};
    void add_demand(int u, ll d) { demand[u] += d; }
    int add(int u, int v, ll l, ll r) {
        demand[u] -= l; demand[v] += l;
        dn.add(u, v, r - l);
        low.push_back(l);
        edge.push_back({ u, sz(dn.graph[u])-1 });
        return sz(edge)-1;
    }
    ll solve() {
        ll sum = 0;
        for (int i = 1; i <= n; i++) sum += demand[i];
        if (sum != 0) return false;
        ll res = 0;
        for (int i = 1; i <= n; i++) {
            if (demand[i] > 0) {
                dn.add(S, i, demand[i]);
                res += demand[i];
            }
            else if (demand[i] < 0) {
                dn.add(i, T, -demand[i]);
            }
        }
        ll f = dn.flow(S, T);
        return (f != res ? -1 : f);
    }
    ll get_flow(int i) { // get actual flow
        auto [u, idx] = edge[i];
        int v = dn.graph[u][idx].to, rev = dn.graph[u][idx].rev;
        return dn.graph[v][rev].cap + low[i];
    }
};
```

4.9 SCC

Usage: Strongly Connected Components (Tarjan's or Kosaraju's).
Time Complexity: $O(V+E)$

```
int N, M, C[10101]; // C[i] = i번 정점이 속한 SCC 번호
vector<int> G[10101], R[10101], V;
vector<vector<int>> S; // 각 SCC에 속한 정점 목록
void AddEdge(int s, int e){
    G[s].push_back(e);
    R[e].push_back(s);
}
void DFS1(int v){
    C[v] = -1;
    for(auto i : G[v]) if(!C[i]) DFS1(i);
    V.push_back(v);
}
void DFS2(int v, int c){
    C[v] = c; S.back().push_back(v);
    for(auto i : R[v]) if(C[i] == -1) DFS2(i, c);
}
int GetSCC(){ // SCC 개수 반환
    for(int i=1; i<=N; i++) if(!C[i]) DFS1(i);
    reverse(V.begin(), V.end());
    int cnt = 0;
    for(auto i : V) if(C[i] == -1) S.emplace_back(), DFS2(i, cnt++);
    return cnt;
} // 각 SCC는 위상 정렬 순서대로 번호 매겨져 있음
```

4.10 BCC

Usage: Biconnected Components, Cut-vertices, and Bridges.
Time Complexity: $O(V+E)$

```
// 1-based, 다른 거 호출하기 전에 tarjan 먼저 호출해야 함
vector<int> G[MAX_V]; int In[MAX_V], Low[MAX_V], P[MAX_V];
void addEdge(int s, int e){ G[s].push_back(e); G[e].push_back(s); }
void tarjan(int n){ /// Pre-Process
    int pv = 0;
    function<void(int,int)> dfs = [&pv,&dfs](int v, int b){
        In[v] = Low[v] = ++pv; P[v] = b;
        for(auto i : G[v]){
            if(i == b) continue;
            if(!In[i]) dfs(i, v), Low[v] = min(Low[v], Low[i]);
            else Low[v] = min(Low[v], In[i]);
        }
    };
    for(int i=1; i<=n; i++) if(!In[i]) dfs(i, -1);
}
vector<int> cutVertex(int n){
    vector<int> res; array<char,MAX_V> isCut; isCut.fill(0);
    function<void(int)> dfs = [&dfs,&isCut](int v){
        int ch = 0;
        for(auto i : G[v]){
            if(P[i] != v) continue; dfs(i); ch++;
            if(P[v] == -1 && ch > 1) isCut[v] = 1;
            else if(P[v] != -1 && Low[i] >= In[v]) isCut[v]=1;
        }
    };
    for(int i=1; i<=n; i++) if(P[i] == -1) dfs(i);
    for(int i=1; i<=n; i++) if(isCut[i]) res.push_back(i);
    return move(res);
}
vector<PII> cutEdge(int n){
    vector<PII> res;
    function<void(int)> dfs = [&dfs,&res](int v){
        for(int t=0; t<G[v].size(); t++){
            int i = G[v][t]; if(t != 0 && G[v][t-1] == G[v][t]) continue;
            if(P[i] != v) continue; dfs(i);
            if((t+1 == G[v].size() || i != G[v][t+1]) && Low[i] > In[v])
            res.emplace_back(min(v,i), max(v,i));
        }
    };
    for(int i=1; i<=n; i++) sort(G[i].begin(), G[i].end()); // multi edge -> sort
    for(int i=1; i<=n; i++) if(P[i] == -1) dfs(i);
    return move(res); // sort(all(res));
}
vector<int> BCC[MAX_V]; // BCC[v] = components which contains v
void vertexDisjointBCC(int n){ // allow multi edge, not allow self loop
    int cnt = 0; array<char,MAX_V> vis; vis.fill(0);
    function<void(int,int)> dfs = [&dfs,&vis,&cnt](int v, int c){
        vis[v] = 1; if(c > 0) BCC[v].push_back(c);
        for(auto i : G[v]){
            if(vis[i]) continue;
            if(In[v] <= Low[i]) BCC[v].push_back(++cnt), dfs(i, cnt);
            else dfs(i, c);
        }
    };
    for(int i=1; i<=n; i++) if(!vis[i]) dfs(i, 0);
    for(int i=1; i<=n; i++) if(BCC[i].empty()) BCC[i].push_back(++cnt);
}
```

5 DP Optimization

5.1 Convex Hull Trick

Usage: `dp[i] = min(dp[j] + b[j] * a[i]), b[j] >= b[j+1]`
Time Complexity: $O(N \log N)$

```
// O(logN) Dynamic CHT: Slopes(k) and queries(x) can be in any order (no sorting
required)
typedef long long ll;
struct Line {
    mutable ll k, m, p;
    bool operator<(const Line& o) const { return k < o.k; }
    bool operator<(ll x) const { return p < x; }
};
struct LineContainer : multiset<Line, less<>> {
    // (for doubles, use inf = 1/.0, div(a,b) = a/b)
    static const ll inf = LLONG_MAX;
    ll div(ll a, ll b) { // floored division
        return a / b - ((a ^ b) < 0 && a % b); }
    bool isect(iterator x, iterator y) {
        if (y == end()) return x->p = inf, 0;
        if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
        else x->p = div(y->m - x->m, x->k - y->k);
        return x->p >= y->p;
    }
    void add(ll k, ll m) { // y = kx + m
        auto z = insert({k, m, 0}), y = z++, x = y;
        while (isect(y, z)) z = erase(z);
        if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
        while ((y = x) != begin() && (--x)->p >= y->p)
            isect(x, erase(y));
    }
    ll query(ll x) {
        assert(!empty());
        auto l = *lower_bound(x);
        return l.k * x + l.m;
    }
} CHT; // add(-k, -m), -query(x) for Lower hull(min)
int main() {
    dp[0] = 0; CHT.add(a[0], dp[0]);
    for (int i = 1; i < n; i++) { // dp[i] = Max j<i(a[j]*b[i] + dp[j])
        dp[i] = CHT.query(b[i]);
        CHT.add(a[i], dp[i]);
    }
    cout << dp[n-1] << "\n";
}
```

5.2 Linear CHT

Usage: CHT when slopes/queries are monotonic.
Time Complexity: $O(N)$

```
// O(1) CHT: Both slopes (k) and queries (x) must be monotonic (sorted).
typedef long long ll;
struct PLL {
    ll x, y;
    PLL(const ll x = 0, const ll y = 0) : x(x), y(y) {}
    bool operator<= (const PLL& i) const { return 1. * x / y <= 1. * i.x / i.y; }
};
struct ConvexHull {
    static ll F(const PLL& i, const ll x) { return i.x * x + i.y; }
    static PLL C(const PLL& a, const PLL& b) { return { a.y - b.y, b.x - a.x }; }
    deque<PLL> S;
    void add(const ll a, const ll b) {
        while (S.size() > 1 && C(S.back(), PLL(a, b)) <= C(S[S.size() - 2],
        S.back())) S.pop_back();
        S.push_back(PLL(a, b));
        /* when x is monotonic decreasing
        while (S.size() > 1 && C(S[0], S[1]) <= C(PLL(a, b), S[0])) S.pop_front()
        S.push_front(PLL(a, b)); */
    }
    ll query(const ll x) {
        while (S.size() > 1 && F(S[0], x) <= F(S[1], x)) S.pop_front(); // upper
        hull(max)
        // while (S.size() > 1 && F(S[0], x) >= F(S[1], x)) S.pop_front(); // lower
        hull(min)
        return F(S[0], x);
    }
} CHT;
int main() { // sorted a, b
    dp[0] = 0; CHT.add(a[0], dp[0]);
    for (int i = 1; i < n; i++) { // dp[i] = Max j<i(a[j]*b[i] + dp[j])
        dp[i] = CHT.query(b[i]);
        CHT.add(a[i], dp[i]);
    }
    cout << dp[n-1] << "\n";
}
```

5.3 D&C optimization

Usage: `dp[t][i] = min(dp[t-1][j] + c[j][i]), c is Monge`
Time Complexity: $O(KN \log N)$

```
ll dp[MAX_K][MAX_N];
// i부터 j까지 구간의 비용을 계산하는 함수 (문제에 맞게 구현)
ll get_cost(int l, int r) { /* return sum[l][r] + C; */ }
// k: 현재 단계(구간 개수 등), pL, pR: 최적의 j를 찾을 탐색 범위
void dnc(int k, int l, int r, int pL, int pR) {
    if (l > r) return;
    int opt = pL, mid = (l + r) / 2;
    dp[k][mid] = -1e18 // 최솟값 문제면 INF, 최댓값 문제면 -INF
    for (int j = pL; j <= min(mid, pR); j++) {
        ll val = (j == 0 ? 0 : dp[k - 1][j - 1]) + get_cost(j, mid);
        if (val > dp[k][mid]) {
            dp[k][mid] = val;
            opt = j;
        }
    }
    dnc(k, l, mid - 1, pL, opt);
    dnc(k, mid + 1, r, opt, pR);
}
// usage: for (int i = 1; i <= T; i++) dnc(i, 0, n-1, 0, n-1);
```

5.4 Monotone Queue optimization

Usage: `dp[i] = min(dp[j] + c[j][i]), c is Monge, find cross`
Time Complexity: $O(N \log N)$

```
// f(j, i) : j에서 i로 전이할 때의 값 (dp[j] + cost(j, i))
ll f(int j, int i);

void solve() {
    auto cross = [&](ll p, ll q) {
        ll lo = max(p, q), hi = n + 1;
        while (lo + 1 < hi) {
            ll mid = (lo + hi) / 2;
            if (f(p, mid) > f(q, mid)) hi = mid; // min 기준: f(p) > f(q)면 q가 우세
            else lo = mid;
        }
        return hi;
    };
    deque<pair<ll,ll>> dq; // {candidate_index, start_pos}
    dq.push_back({0, 1}); // 초기값: 0번이 1번 위치부터 최적이라고 가정
    for (int i = 1; i <= n; i++) {
        while (dq.size() > 1 && dq[1].second <= i) dq.pop_front();
        dp[i] = f(dq[0].first, i);
        while (!dq.empty()) {
            ll p = dq.back().first;
            ll pos = cross(p, i);
            if (pos <= dq.back().second) dq.pop_back();
            else {
                if (pos <= n) dq.push_back({i, pos});
                break;
            }
        }
        if (dq.empty()) dq.push_back({i, 1});
    }
}
```

5.5 Aliens Trick

Usage: dp[t][i] = min(dp[t-1][j] + c[j+1][i]), c is Monge, find lambda w/ half bs
Time Complexity: $O(T \log X)$

```
/**
 * n: 원소 개수 (경로 복원 끝점), k: 정확히 골라야 하는 개수
 * lo, hi: 패널티 이분탐색 범위 (0 ~ 최대 가치)
 * f(c): 패널티가 c일 때 {2*(가치합), prv} 반환 (가치는 c를 뺀 값)
 */
template<class T, bool GET_MAX = false>
pair<T, vector<int>> AliensTrick(int n, int k, auto f, T lo, T hi) {
    T l = lo, r = hi;
    while (l < r) {
        T m = (l + r + (GET_MAX ? 1 : 0)) >> 1;
        vector<int> prv = f(m * 2 + (GET_MAX ? -1 : 1)).second;
        int cnt = 0; for (int i = n; i; i = prv[i]) cnt++;
        if (cnt <= k) (GET_MAX ? l : r) = m;
        else (GET_MAX ? r : l) = m + (GET_MAX ? -1 : 1);
    }
    T opt_val = f(l * 2).first / 2 - k * l;
    auto get_path = [&](T c) {
        vector<int> p{n};
        for (auto prv = f(c).second; p.back(); ) p.push_back(prv[p.back()]);
        reverse(p.begin(), p.end()); return p;
    };
    auto p1 = get_path(l * 2 + (GET_MAX ? 1 : -1));
    auto p2 = get_path(l * 2 - (GET_MAX ? 1 : -1));
    if (p1.size() == k + 1) return {opt_val, p1};
    if (p2.size() == k + 1) return {opt_val, p2};
    for (int i = 1, j = 1; i < p1.size(); i++) {
        while (j < p2.size() && p2[j] < p1[i - 1]) j++;
        if (p1[i] <= p2[j] && i - j == k + 1 - (int)p2.size()) {
            vector<int> res(p1.begin(), p1.begin() + i);
            res.insert(res.end(), p2.begin() + j, p2.end());
            return {opt_val, res};
        }
    }
    return {opt_val, {}}; // Should not reach here
}
```

5.6 Sum Over Subsets

Usage: dp[mask] = sum(A[i]), i is in mask
Time Complexity: $O(N2^N)$

```
for (int i = 0; i < (1<<n); i++)
    f[i] = a[i];
for (int j = 0; j < n; j++)
    for(int i = 0; i < (1<<n); i++)
      if (i & (1<<j)) f[i] += f[i ^ (1<<j)];
```

6 Geometry

6.1 Geometry Template

Usage: Basic point, line, and circle operations.

```
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;
using ll = long long;
using Point = pair<ll, ll>;

Point operator + (Point p1, Point p2){ return {p1.x + p2.x, p1.y + p2.y}; }
Point operator - (Point p1, Point p2){ return {p1.x - p2.x, p1.y - p2.y}; }
ll operator * (Point p1, Point p2){ return p1.x * p2.x + p1.y * p2.y; } // 내적
ll operator / (Point p1, Point p2){ return p1.x * p2.y - p2.x * p1.y; } // 외적
int Sign(ll v){ return (v > 0) - (v < 0); } // 양수면 +1, 음수면 -1, 0이면 0 반환
ll Dist(Point p1, Point p2){ return (p2 - p1) * (p2 - p1); } // 두 점 거리 제곱
ll SignedArea(Point p1, Point p2, Point p3){ return (p2 - p1) / (p3 - p1); }
int CCW(Point p1, Point p2, Point p3){ return Sign(SignedArea(p1, p2, p3)); }
```

6.2 Convex Hull

Usage: Monotone Chain or Graham Scan.
Time Complexity: $O(N \log N)$

```
// 모든 점을 포함하는 가장 작은 볼록 다각형, O(N log N)
vector<Point> ConvexHull(vector<Point> points){ // Graham scan
    if(points.size() <= 1) return points;
    swap(points[0], *min_element(all(points)));
    sort(points.begin()+1, points.end(), [&](auto a, auto b){
        int dir = CCW(points[0], a, b);
        if(dir != 0) return dir > 0;
        return Dist(points[0], a) < Dist(points[0], b);
    });
    vector<Point> hull;
    for(auto p : points){
        while(hull.size() >= 2 && CCW(hull[hull.size()-2], hull.back(), p) <= 0)
        hull.pop_back();
        hull.push_back(p);
    }
    return hull;
}
vector<Point> convexHull(vector<Point> points) { // Monotone chain
    if (sz(points) <= 1) return points;
    sort(all(points), [&](Point p1, Point p2) {
        return p1.x == p2.x ? p1.y < p2.y : p1.x < p2.x;
    });
    Polygon v(sz(points)+1);
    int s = 0, t = 0;
    for (int i = 2; i--; s = --t, reverse(all(points))) {
        for (auto p : points) {
            while (t >= s+2 && ccw(v[t-2], v[t-1], p) <= 0) t--;
            v[t++] = p;
        }
    }
    v.resize(t - (t==2 && v[0].x == v[1].x && v[0].y == v[1].y));
    return v;
}
```

6.3 Rotating Calipers

Usage: Maximum distance/diameter of a convex hull.
Time Complexity: $O(N)$

```
// 가장 먼 두 점을 구하는 함수, O(N)
// 주의: hull은 반시계 방향으로 정렬된 볼록 다각형이어야 함
pair<Point, Point> Calipers(vector<Point> hull){
    int n = hull.size(); ll mx = 0; Point a, b;
    for(int i=0, j=0; i<n; i++){
        while(j + 1 < n && (hull[i+1] - hull[i]) / (hull[j+1] - hull[j]) >= 0){
            ll now = Dist(hull[i], hull[j]);
            if(now > mx) mx = now, a = hull[i], b = hull[j];
            j++;
        }
        ll now = Dist(hull[i], hull[j]);
        if(now > mx) mx = now, a = hull[i], b = hull[j];
    }
    return {a, b};
}
```

6.4 Point in Convex Polygon

Usage: Binary search based inclusion test.
Time Complexity: $O(\log N)$

```
// 다각형 내부 또는 경계 위에 p가 있으면 true, O(log N)
// 주의: v는 반시계 방향으로 정렬된 볼록 다각형이어야 함
bool PointInConvexPolygon(const vector<Point> &v, const Point &pt){
    if(CCW(v[0], v[1], pt) < 0) return false; int l = 1, r = v.size() - 1;
    while(l < r){
        int m = l + r + 1 >> 1;
        if(CCW(v[0], v[m], pt) >= 0) l = m; else r = m - 1;
    }
    if(l == v.size() - 1) return CCW(v[0], v.back(), pt) == 0 && v[0] <= pt && pt <=
    v.back();
    return CCW(v[0], v[l], pt) >= 0 && CCW(v[l], v[l+1], pt) >= 0 && CCW(v[l+1],
    v[0], pt) >= 0;
}
```

6.5 Point in Polygon

Usage: Ray casting algorithm for general polygons.
Time Complexity: $O(N)$

```
// 다각형 내부 또는 경계 위에 p가 있으면 true, O(N)
bool PointInPolygon(const vector<Point> &v, Point p){
    int n = v.size(), cnt = 0;
    Point p2(p.x+1, 1'000'000'000 + 1); // 좌표 범위보다 큰 수
    for(int i=0; i<n; i++){
        int j = i + 1 < n ? i + 1 : 0;
        if(min(v[i],v[j]) <= p && p <= max(v[i],v[j]) && CCW(v[i], v[j], p) == 0)
        return true;
        if(SegmentIntersection(v[i], v[j], p, p2)) cnt++;
    }
    return cnt % 2 == 1;
}
```

6.6 Polar Sort

Usage: Sorting points by angle.
Time Complexity: $O(N \log N)$

```
/* y축
    ↑
3  2 1
4 -1 0 → x축
5  6 7
원점 기준 각도 정렬
x축 위에 있는 점이 가장 먼저, 그리고 반시계 방향으로
*/
int QuadrantID(const Point p){
  static int arr[9] = { 5, 4, 3, 6, -1, 2, 7, 0, 1 };
  return arr[Sign(p.x)*3+Sign(p.y)+4];
}
sort(points.begin(), points.end(), [&](auto p1, auto p2){
  if(QuadrantID(p1) != QuadrantID(p2)) return QuadrantID(p1) < QuadrantID(p2);
  else return p1 / p2 > 0; // 반시계
});
```

6.7 Polygon Area

Usage: Shoelace formula.
Time Complexity: $O(N)$

```
// 다각형의 넓이의 2배를 반환, 항상 정수, O(N)
ll PolygonArea(const vector<Point> &v){
    ll res = 0;
    for(int i=0; i<v.size(); i++) res += v[i] / v[(i+1)%v.size()];
    return abs(res);
}
```

6.8 Segment Intersection

Usage: Check if two line segments intersect.
Time Complexity: $O(1)$

```
// 선분 교차 - 선분 ab와 선분 cd가 만나면 true
bool Cross(Point s1, Point e1, Point s2, Point e2){
    int ab = CCW(s1, e1, s2) * CCW(s1, e1, e2);
    int cd = CCW(s2, e2, s1) * CCW(s2, e2, e1);
    if(ab == 0 && cd == 0){
        if(s1 > e1) swap(s1, e1);
        if(s2 > e2) swap(s2, e2);
        return !(e1 < s2 || e2 < s1);
    }
    return ab <= 0 && cd <= 0;
}
// 교차하지 않으면 0
// 교점이 무한히 많으면 -1
// 교점이 1개면 1 반환하고 res에 교점 저장
int Cross(Point s1, Point e1, Point s2, Point e2, pair<double, double> &res){
    if(!Cross(s1, e1, s2, e2)) return 0;
    ll det = (e1 - s1) / (e2 - s2);
    if(!det){
        if(s1 > e1) swap(s1, e1);
        if(s2 > e2) swap(s2, e2);
        if(e1 == s2){ res = s2; return 1; }
        if(e2 == s1){ res = s1; return 1; }
        return -1;
    }
    res.x = s1.x + (e1.x - s1.x) * ((s2 - s1) / (e2 - s2) * 1.0 / det);
    res.y = s1.y + (e1.y - s1.y) * ((s2 - s1) / (e2 - s2) * 1.0 / det);
    return 1;
}
```

7 String

7.1 Aho-Corasick

Usage: Multi-pattern matching using trie and failure links.
Time Complexity: $O(\sum |P| + |T|)$

```
struct AhoCorasick {
    struct Node {
        Node *nxt[26], *fail;
        vector<int> out; // 패턴의 인덱스 저장
        int terminal;
        Node() : fail(nullptr), terminal(-1) { fill(nxt, nxt + 26, nullptr); }
        ~Node() {
            for (int i = 0; i < 26; i++) if (nxt[i]) delete nxt[i];
        }
        void insert(const char* s, int id) {
            if (*s == 0) { terminal = id; out.push_back(id); return; }
            int curr = *s - 'a';
            if (!nxt[curr]) nxt[curr] = new Node();
            nxt[curr]->insert(s + 1, id);
        }
    };
    Node* root;
    AhoCorasick() { root = new Node(); }
    ~AhoCorasick() { delete root; }
    void insert(const string& s, int id) { root->insert(s.c_str(), id); }
    void build() {
        queue<Node*> q;
        root->fail = root;
        for (int i = 0; i < 26; i++) {
            if (root->nxt[i]) {
                root->nxt[i]->fail = root;
                q.push(root->nxt[i]);
            } else {
                root->nxt[i] = root; // DFA optimization
            }
        }
        while (!q.empty()) {
            Node* curr = q.front(); q.pop();
            for (int i = 0; i < 26; i++) {
                if (curr->nxt[i]) {
                    Node* next = curr->nxt[i];
                    next->fail = curr->fail->nxt[i];
                    next->out.insert(next->out.end(), next->fail->out.begin(),
                    next->fail->out.end());
                    q.push(next);
                } else {
                    curr->nxt[i] = curr->fail->nxt[i]; // DFA optimization
                }
            }
        }
    }
    vector<pair<int, int>> query(const string& s) {
        vector<pair<int, int>> res;
        Node* curr = root;
        for (int i = 0; i < s.size(); i++) {
            curr = curr->nxt[s[i] - 'a'];
            for (int id : curr->out) res.emplace_back(i, id);
        }
        return res;
    }
};
int main() {
    AhoCorasick ac;
    vector<string> patterns = {"he", "she", "hers", "his"};
    for(int i = 0; i < patterns.size(); i++)
        ac.insert(patterns[i], i); // 패턴과 ID(0~N-1) 삽입
    ac.build(); // 실패 함수/DFA 빌드 (필수)
    string text = "ushers";
    auto res = ac.query(text); // 탐색: {끝 인덱스, 패턴 ID} 쌍 반환
    for (auto& [idx, id] : res) {
        // patterns[id] 가 text의 idx에서 끝남을 의미
    }
}
```

7.2 Hashing

Usage: Rolling hash for string matching.
Time Complexity: $O(N)$

```
// 전처리 O(N), 부분 문자열의 해시값을 O(1)에 구함
// Hashing<917, 998244353> H;
// H.build("ABCDABCD");
// assert(H.get(1, 4) == H.get(5, 8));
// 주의: get 함수의 인자는 1-based 닫힌 구간
// 주의: M은 10억 근처의 소수, P는 M과 서로소

// 1e5+3, 1e5+13, 131'071, 524'287, 1'299'709, 1'301'021
// 1e9-63, 1e9+7, 1e9+9, 1e9+103
template<long long P, long long M> struct Hashing {
    vector<long long> h, p;
    void build(const string &s){
        int n = s.size();
        h = p = vector<long long>(n+1); p[0] = 1;
        for(int i=1; i<=n; i++) h[i] = (h[i-1] * P + s[i-1]) % M;
        for(int i=1; i<=n; i++) p[i] = p[i-1] * P % M;
    }
    long long get(int s, int e) const {
        long long res = (h[e] - h[s-1] * p[e-s+1]) % M;
        return res >= 0 ? res : res + M;
    }
};
```

7.3 KMP

Usage: Single pattern matching using prefix function.
Time Complexity: $O(N + M)$

```
// s에서 p가 등장하는 위치 반환
// KMP("ABABCAB", "AB") = {0, 2, 5}
// KMP("AAAA", "AA") = {0, 1, 2}

vector<int> GetFail(const string &p){
    int n = p.size();
    vector<int> fail(n);
    for(int i=1, j=0; i<n; i++){
        while(j && p[i] != p[j]) j = fail[j-1];
        if(p[i] == p[j]) fail[i] = ++j;
    }
    return fail;
}

vector<int> KMP(const string &s, const string &p){
    int n = s.size(), m = p.size();
    vector<int> fail = GetFail(p), ret;
    for(int i=0, j=0; i<s.size(); i++){
        while(j && s[i] != p[j]) j = fail[j-1];
        if(s[i] == p[j]){
            if(j + 1 == m) ret.push_back(i-m+1), j = fail[j];
            else j++;
        }
    }
    return ret;
}
```

7.4 Manacher

Usage: Find all palindromic substrings in linear time.
Time Complexity: $O(N)$

```
// 각 문자를 중심으로 하는 최장 팰린드롬의 반경을 반환
// Manacher("abaaba") = {0,1,0,3,0,1,6,1,0,3,0,1,0}
// # a # b # a # a # b # a #
// 0 1 0 3 0 1 6 1 0 3 0 1 0
vector<int> Manacher(const string &inp){
    int n = inp.size() * 2 + 1;
    vector<int> ret(n);
    string s = "#";
    for(auto i : inp) s += i, s += "#";
    for(int i=0, p=-1, r=-1; i<n; i++){
        ret[i] = i <= r ? min(r-i, ret[2*p-i]) : 0;
        while(i-ret[i]-1 >= 0 && i+ret[i]+1 < n && s[i-ret[i]-1] == s[i+ret[i]+1])
        ret[i]++;
        if(i+ret[i] > r) r = i+ret[i], p = i;
    }
    return ret;
}
```

7.5 Suffix Array

Usage: Suffix array and LCP array construction.
Time Complexity: $O(N \log N)$

```
// LCP는 1-based
pair<vector<int>, vector<int>> SuffixArray(const string &s){ // O(N log N)
    int n = s.size(), m = max(n, 256);
    vector<int> sa(n), lcp(n), pos(n), tmp(n), cnt(m);
    auto counting_sort = [&](){
        fill(cnt.begin(), cnt.end(), 0);
        for(int i=0; i<n; i++) cnt[pos[i]]++;
        partial_sum(cnt.begin(), cnt.end(), cnt.begin());
        for(int i=n-1; i>=0; i--) sa[--cnt[pos[tmp[i]]]] = tmp[i];
    };
    for(int i=0; i<n; i++) sa[i] = i, pos[i] = s[i], tmp[i] = i;
    counting_sort();
    for(int k=1; ; k<<=1){
        int p = 0;
        for(int i=n-k; i<n; i++) tmp[p++] = i;
        for(int i=0; i<n; i++) if(sa[i] >= k) tmp[p++] = sa[i] - k;
        counting_sort();
        tmp[sa[0]] = 0;
        for(int i=1; i<n; i++){
            tmp[sa[i]] = tmp[sa[i-1]];
            if(sa[i-1]+k < n && sa[i]+k < n && pos[sa[i-1]] == pos[sa[i]] &&
            pos[sa[i-1]+k] == pos[sa[i]+k]) continue;
            tmp[sa[i]] += 1;
        }
        swap(pos, tmp); if(pos[sa.back()] + 1 == n) break;
    }
    for(int i=0, j=0; i<n; i++, j=max(j-1,0)){
        if(pos[i] == 0) continue;
        while(sa[pos[i]-1]+j < n && sa[pos[i]]+j < n && s[sa[pos[i]-1]+j] ==
        s[sa[pos[i]]+j]) j++;
        lcp[pos[i]] = j;
    }
    return {sa, lcp};
}
```

7.6 Z-algorithm

Usage: Longest common prefix between S and its suffixes.
Time Complexity: $O(N)$

```
// Z[i] = LongestCommonPrefix(S[0:N], S[i:N])
//      = S[0:N]과 S[i:N]이 앞에서부터 몇 글자 겹치는지
vector<int> Z(const string &s){
    int n = s.size();
    vector<int> z(n);
    z[0] = n;
    for(int i=1, l=0, r=0; i<n; i++){
        if(i < r) z[i] = min(r-i-1, z[i-l]);
        while(i+z[i] < n && s[i+z[i]] == s[z[i]]) z[i]++;
        if(i+z[i] > r) r = i+z[i], l = i;
    }
    return z;
}
```

8 STL & pbds

8.1 Hash map

Usage: Faster hash table using pb_ds.
Time Complexity: $O(1)$

```
// faster than unordered_map
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;

gp_hash_table<int,int> hashmap;
// cannot use hashmap.count()
```

8.2 Ordered Set

Usage: Set supporting order_of_key and find_by_order.
Time Complexity: $O(\log N)$

```
// k번째 원소 확인(find_by_order) 및 x보다 작은 원소 개수 확인(order_of_key)을
O(logN)에 수행
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;
// ordered_set<int> os;
// os.order_of_key(x)  : x보다 작은 원소의 개수 반환
// os.find_by_order(k) : k번째 원소의 iterator 반환 (0-indexed, 없으면 OS.end())

template <typename T>
using ordered_multiset = tree<T, null_type, less_equal<T>, rb_tree_tag,
tree_order_statistics_node_update>;
void m_erase(ordered_multiset<int> &os, int val) { // multiset 전용 erase 함수
    int idx = os.order_of_key(val);
    auto it = os.find_by_order(idx);
    if (it != os.end() && *it == val) os.erase(it);
}
auto m_find(ordered_multiset<int> &os, int val) { // multiset 전용 find 함수
    int idx = os.order_of_key(val);
    auto it = os.find_by_order(idx);
    if (it != os.end() && *it == val) return it;
    return os.end();
}
```

8.3 Permutation & Combination

Usage: next_permutation and mask-based combinations.

```
#include <algorithm>
/* 1. Permutation */
sort(all(v))
do {
    // process v
} while (next_permutation(all(v)));

/* 2. Combination (nCr): Use a mask vector */
vector<int> mask(n, 0);
fill(mask.end()-r, mask.end(), 1); // pick r elements
do {
    for (int i = 0; i < n; i++) {
        if (mask[i]) { /* v[i] is selected */ }
    }
} while (next_permutation(all(mask)));

/* 3. Partial Permutation (nPk) */
```

```
sort(all(v));
do {
    for(int i = 0; i < k; i++) { /* use v[i] */ }
    reverse(v.begin()+k, v.end());
} while (next_permutation(all(v)));
```

8.4 Priority Queue

Usage: Meldable/Thin heaps from pb_ds.
Time Complexity: $O(\log N)$

```
// 큐 병합, 임의 값 수정 및 삭제 가능
#include <functional>
#include <ext/pb_ds/priority_queue.hpp>
using namespace __gnu_pbds;
template <typename T>
using pbds_pq = __gnu_pbds::priority_queue<T, less<T>, pairing_heap_tag>;
int main() {
    pbds_pq<int> pq1, pq2;
    auto it = pq1.push(10);
    pq2.push(100);
    pq1.join(pq2); // O(1), pq1: {10, 100}, pq2: {}
    pq1.modify(it, 50); // O(logN), pq1 : {50, 100}
    pq1.erase(it); // O(logN), pq1: {100}
    pq1.top(); pq1.empty(); pq1.size(); pq1.pop(); // same
}
```

8.5 Rope

Usage: Fast insertions/deletions in large strings.
Time Complexity: $O(\log N)$

```
#include<ext/rope>
using namespace __gnu_cxx;
int main() {
    string str; cin >> str;
    crope r(str.c_str());
    // vector<T> v; rope<T> r(v);
    r.insert(pos, str); // Insert O(logN)
    r.erase(pos, len); // Erase O(logN)
    r.replace(pos, len, str); // Replace O(logN)
    crope r2 = r; // O(1)
    r2 = r.substr(pos, len); // O(logN + len)
    r += r2; // Append O(logN + len(r2))
    r[idx]; // O(logN)
    for (auto i : r) // Traversal O(N)?
    cout << r;
}
```

8.6 Trie

Usage: Prefix tree implementation from pb_ds.

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/trie_policy.hpp>
using namespace __gnu_pbds;
typedef trie<string, null_type, trie_string_access_traits<>, pat_trie_tag,
trie_prefix_search_node_update> trie_set;

int main() {
    trie_set t; t.insert("apple");
    t.insert("app"); t.insert("banana");
    if (t.find("app") != t.end()) { /* found app */ }
    auto [st, en] = t.prefix_range("ba");
    for (auto it = st; it != en; it++) {
        cout << *it << "\n"; // banana
    }
    *t.lower_bound("app"); // app
    *t.upper_bound("app"); // apple
    t.split("b", t2); // t: {app, apple}, t2: {banana}
    t.erase("apple");
}
```

9 Misc

9.1 Custom Hash

Usage: Custom hash for pair, vector.

```
struct custom_hash {
    template <class T>
    void combine(size_t& seed, const T& v) const {
        seed ^= hash<T>{}(v) + 0x9e3779b9 + (seed << 6) + (seed >> 2);
    }
    template <class T1, class T2>
    size_t operator()(const pair<T1, T2>& p) const {
        size_t seed = 0;
        combine(seed, p.first);
        combine(seed, p.second);
        return seed;
    }
    template <class T>
    size_t operator()(const vector<T>& v) const {
        size_t seed = 0;
        for (const auto& i : v) combine(seed, i);
        return seed;
    }
};
```

9.2 Fast I/O

Usage: Fast integer I/O using fread/fwrite.

```
// #define fastio cin.tie(0)->sync_with_stdio(0) // for normal
```

```cpp
#include <unistd.h>
constexpr int rbuf_sz = 1 << 20;
constexpr int wbuf_sz = 1 << 20;
int main() {
  char r[rbuf_sz], *pr = r; read(0, r, rbuf_sz);
  auto read_char = [&] {
    if (pr - r == rbuf_sz) read(0, pr = r, rbuf_sz);
    return *pr++;
  };
  auto read_int = [&] {
    int ret = 0, flag = 0;
    char c = read_char();
    while (c == ' ' || c == '\n') c = read_char();
    if (c == '-') flag = 1, c = read_char();
    while (c != ' ' && c != '\n') ret = 10 * ret + c - '0', c = read_char();
    if (flag) ret = -ret;
    return ret;
  };
    char w[wbuf_sz], *pw = w;
  auto write_char = [&](char c) {
    if (pw - w == wbuf_sz) write(1, w, pw - w), pw = w;
    *pw++ = c;
  };
  auto write_int = [&](int x) {
    if (pw - w + 40 > wbuf_sz) write(1, w, pw - w), pw = w;
    if (x < 0) *pw++ = '-', x = -x;
    char t[10], *pt = t;
    do *pt++ = x % 10 + '0'; while (x /= 10);
    do *pw++ = *--pt; while (pt != t);
  };
}
```

9.3 Random

Usage: Better random for mt19937.

```cpp
#include <random>
#include <chrono>
using namespace std;
mt19937_64 rng(chrono::high_resolution_clock::now().time_since_epoch().count());

uniform_int_distribution<int>(l, r)(rng); // [l, r]
uniform_real_distribution<double>(l, r)(rng); // [l, r)
shuffle(all(v), rng) // shuffle vector
vector<double> w = { 40, 10, 50 };
discrete_distribution<int>(all(w))(rng); // 0: 40%, 1: 10%, 2: 50%
bernoulli_distribution(P)(rng); // True with probability P(0.0~1.0)
```

9.4 Ternary Search

Usage: Finding extremum of unimodal functions.
Time Complexity: $O(\log N)$

```cpp
typedef long long ll;
ll ternary_search(ll lo, ll hi, auto f) {
    while (hi - lo >= 3) {
        ll p = lo + (hi - lo) / 3, q = hi - (hi - lo) / 3;
        if (f(p) < f(q)) hi = q;
        else lo = p;
    }
    ll idx = lo; auto minv = f(lo);
    for (ll i = lo+1; i <= hi; i++) {
        auto v = f(i);
        if (v < minv) {
            minv = v; idx = i;
        }
    }
    return idx;
}
double ternary_search(double lo, double hi, auto f, int iter = 100) {
    for (int i = 0; i < iter; i++) {
        double p = (lo*2 + hi) / 3., q = (lo + hi*2) / 3.;
        if (f(p) < f(q)) hi = q;
        else lo = p;
    }
    return (lo+hi) / 2.;
}
```

10 Checklist + Useful Info

10.1 Useful Stuff

- Catalan Number
  1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900
  $C_n = \binom{2n}{n}/(n+1);$
  - 길이가 2n인 올바른 괄호 수식의 수
  - n + 1개의 리프를 가진 풀 바이너리 트리의 수
  - n + 2각형을 n개의 삼각형으로 나누는 방법의 수

- Burnside’ s Lemma
  경우의 수를 세는데, 특정 transform operation(회전, 반사, ..) 해서 같은 경우들은 하나로 친다. 전체 경우의 수는? 각 operation마다 이 operation을 했을 때 변하지 않는 경우의 수를 센다 (단, “아무것도 하지 않는다” 라는 operation도 있어야 함!) 전체 경우의 수를 더한 후, operation의 수로 나눈다. (답이 맞다면 항상 나누어 떨어져야 한다)

- 알고리즘 게임
  - Nim Game의 해법 : 각 더미의 돌의 개수를 모두 XOR했을 때 0 이 아니면 첫번째, 0 이면 두번째 플레이어가 승리.
  - Grundy Number : 어떤 상황의 Grundy Number는, 가능한 다음 상황들의 Grundy Number를 모두 모은 다음, 그 집합에 포함 되지 않는 가장 작은 수가 현재 state의 Grundy Number가 된다. 만약 다음 state가 독립된 여러개의 state들로 나뉠 경우, 각각의 state의 Grundy Number의 XOR 합을 생각한다.
  - Subtraction Game : 한 번에 k 개까지의 돌만 가져갈 수 있는 경우, 각 더미의 돌의 개수를 k + 1로 나눈 나머지를 XOR 합하여 판단한다.
  - Index-k Nim : 한 번에 최대 k개의 더미를 골라 각각의 더미에서 아무렇게나 돌을 제거할 수 있을

때, 각 binary digit에 대하여 합을 k + 1로 나눈 나머지를 계산한다. 만약 이 나머지가 모든 digit에 대하여 0이라면 두번째, 하나라도 0이 아니라면 첫번째 플레이어가 승리.

- Pick' s Theorem
  격자점으로 구성된 simple polygon이 주어짐. I 는 polygon 내부의 격자점 수, B 는 polygon 선분위 격자점 수, A는 polygon의 넓이라고 할 때, 다음과 같은 식이 성립한다. $A = I + B/2 - 1$
- 가장 가까운 두 점 : 분할정복으로 가까운 6개의 점만 확인
- 홀의 결혼 정리 : 이분그래프(L-R)에서, 모든 L을 매칭하는 필요충분 조건 = L에서 임의의 부분집합 S를 골랐을 때, 반드시 (S의 크기) $\leq$ (S와 연결되어있는 모든 R의 크기)이다.
- 소수 : 10 007 , 10 009 , 10 111 , 31 567 , 70 001 , 1 000 003 , 1 000 033 , 4 000 037 , 99 999 989 , 999 999 937 , 1 000 000 007 , 1 000 000 009 , 9 999 999 967 , 99 999 999 977
- 소수 개수 : (1e5 이하 : 9592), (1e7 이하 : 664 579) , (1e9 이하 : 50 847 534)
- 10^{15} 이하의 정수 범위의 나눗셈 한번은 오차가 없다.
- N의 약수의 개수 = $O(N^{1/3})$, N의 약수의 합 = $O(N \log\log N)$
- $\phi(mn) = \phi(m)\phi(n), \phi(pr^n) = pr^n - pr^{n-1}, a^{\phi(n)} \equiv 1 \pmod{n}$ if coprime
- Euler characteristic : v - e + f (면, 외부 포함) = 1 + c (컴포넌트)
- Euler's phi $\phi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right)$
- Lucas' Theorem $\binom{m}{n} = \prod \binom{m_i}{n_i} \pmod{p}$ m_i, n_i는 p^i의 계수
- 스케줄링에서 데드라인이 빠른 걸 쓰는게 이득. 늦은 스케줄이 안들어갈 때 가장 시간 소모가 큰 스케줄 1개를 제거하면 이득.

10.2 자주 쓰이는 문제 접근법

- 비슷한 문제를 풀어본 적이 있던가?
- 단순한 방법에서 시작할 수 있을까? (brute force)
- 내가 문제를 푸는 과정을 수식화할 수 있을까? (예제를 직접 해결해보면서)
- 문제를 단순화할 수 없을까?
- 그림으로 그려볼 수 있을까?
- 수식으로 표현할 수 있을까?
- 문제를 분해할 수 있을까?
- 뒤에서부터 생각해서 문제를 풀 수 있을까?
- 순서를 강제할 수 있을까?
- 특정 형태의 답만을 고려할 수 있을까? (정규화)
- 특수 조건을 꼭 활용
- 여사건으로 생각하기
- 게임이론 - 거울 전략 혹은 mex DP 연계
- 겁먹지 말고 경우 나누어 생각
- 해법에서 역순으로 가능한가?
- 딱 맞는 시간복잡도에 집착하지 말자
- 문제에 의미있는 작은 상수 이용
- 스몰투라지, 트라이, 해싱, 루트질 같은 트릭 생각
- 너무 추상화하기보단 풀려야 하는 방식으로 생각하기
- 잘못된 방법으로 파고들지 말고 버리자
- 제발 터널 비전에 빠지지 말자
- 헬프 콜은 적극적으로
- 혼자 멘탈 나가지 않기

10.3 DP 최적화 접근

- C[i, j] = A[i] * B[j]이고 A, B가 단조증가, 단조감소이면 Monge
- l..r의 값들의 sum이나 min은 Monge
- 식 정리해서 일차(CHT) 혹은 비슷한(MQ) 함수를 발견, 구현 힘들면 Li-Chao
- $a \leq b \leq c \leq d$에서 $A[a, c] + A[b, d] \leq A[a, d] + A[b, c]$
- Monge 성질을 보이기 어려우면 N^2 나이브 짜서 opt의 단조성을 확인하고 찍맞
- 식이 간단하거나 변수가 독립적이면 DP 테이블을 세그 위에 올려서 해결
- 침착하게 점화식부터 세우고 Monge인지 판별
- Monge에 집착하지 말고 단조성이나 볼록성만 보여도 됨