Team Note of A Team

A, B, C

Compiled on January 12, 2026

Contents

1 Basic Implementation

1.1 Main Template

```
// #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;
typedef unsigned long long ull;
const ll INF = 1e18;
const int MOD = 998244353;
const int SIZE = 524288;
```

2 Math

2.1 Basic Arithmetic

```
ll modmul(ull a, ull b, ull M) {
  ll ret = a * b - M * ull(1.L / M * a * b);
  return ret + M * (ret < 0) - M * (ret >= (ll)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: first: $O(1)$ / second: $O(\sum p^k)$

```
// when M is big prime; Init: O(MAXN), Query: O(1)
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)
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, ll 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);
  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 = [&](ll p, ll pe, ll 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, 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: $O(\log N)$

```
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: $O(N \log N)$

```
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, 1e9+7);
  poly p1({1, 2, 1}), p2({1, 1}); // NTT base
  p1 * p2; p1 / p2; p1 % p2; // polynomial operation
  // ex. A_n = 1*A_{n-1} + 1*A_{n-2}
  poly coef({1, 1}); // {c0, c1} 순서 (A_{n-2}, A_{n-1}
  계수)
  poly init({0, 1}); // {A0, A1} 초기값
  cout << Kitamasa(coef, init, 1e9).v;
}
```

2.5 Linear Sieve

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

```
struct Sieve {
  // sp: 최소 소인수, e: i의 최소 소인수 지수, phi: 오
  일러 피 함수(1~i 중 i와 서로소인 개수), mu: 뫼비우스
  함수, tau: 약수 개수, sigma: 약수의 합
  vector<int> sp, e, phi, mu, tau, sigma, tmp, primes;
  Sieve(int n) : sp(n+1), e(n+1), phi(n+1), mu(n+1),
  tau(n+1), sigma(n+1), tmp(n+1) {
    phi[1] = mu[1] = tau[1] = sigma[1] = 1;
    for (int i = 2; i <= n; i++) {
      if (!sp[i]) {
        sp[i]=i; primes.push_back(i);
```

```
        e[i]=1; phi[i]=i-1; mu[i]=-1; tau[i]=2;
        sigma[i]=tmp[i]=i+1;
      }
      for (int p : primes) {
        if (i*p > n || p > sp[i]) break;
        int m = i*p; sp[m] = p;
        if (i%p == 0) {
          e[m] = e[i]+1; phi[m] = phi[i]*p; mu[m] = 0;
          tau[m] = tau[i]/(e[i]+1)*(e[m]+1);
          tmp[m] = tmp[i]*p+1; sigma[m] =
          sigma[i]/tmp[i]*tmp[m];
          break;
        }
        e[m] = 1; phi[m] = phi[i]*(p-1); mu[m] =
        -mu[i];
        tau[m] = tau[i]*2; tmp[m] = p+1; sigma[m] =
        sigma[i]*(p+1);
      }
    }
  }
};
```

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

```
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

```
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 1D & 2D Fenwick Tree

Usage: Point updates and subgrid sums on a 2D plane.
Time Complexity: 1D: $O(\log N)$, 2D: $O(\log N \log M)$

```
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(); }
};
```

3.4 Merge Sort Tree

Usage: Count/rank of elements in range $[L, R]$.
Time Complexity: $O(\log^2 N)$ per 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(sz(tree[i*2]) + sz(tree[i*2+1]));
      merge(all(tree[i*2]), all(tree[i*2+1]),
      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(all(tree[l]), k); l++;
      }
      if (!(r & 1)) {
        res += tree[r].end() -
        upper_bound(all(tree[r]), 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;
  }
};
```

3.5 Persistent Segment Tree

Usage: Accessing previous versions and range k-th element.
Time Complexity: $O(\log N)$ per 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);
}
```

3.6 Sweepline Mo's

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

```
const int MAXN = 200005, 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); // 필
    요 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 {
```

```cpp
  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;
}
```

3.7 Link-Cut Tree

Usage: Dynamic tree structure supporting link, cut, and path operations using auxiliary Splay trees.
Time Complexity: Amortized $O(\log N)$

```cpp
struct Node {
  Node *l, *r, *p;
  bool flip; int sz;
  T now, sum, lz;
  Node() {
    l = r = p = nullptr; sz = 1;
    flip = false; now = sum = lz = 0;
  }
  bool IsLeft() const { return p && this == p->l; }
  bool IsRoot() const { return !p || (this != p->l &&
  this != p->r); }
  friend int GetSize(const Node *x) { return x ? x->sz
  : 0; }
  friend T GetSum(const Node *x) { return x ? x->sum :
  0; }
  void Rotate() {
    p->Push(); Push();
    if (IsLeft())
      r && (r->p = p), p->l = r, r = p;
    else
      l && (l->p = p), p->r = l, l = p;
    if (!p->IsRoot())
      (p->IsLeft() ? p->p->l : p->p->r) = this;
    auto t = p; p = t->p;
    t->p = this; t->Update();
    Update();
  }
  void Update() {
    sz = 1 + GetSize(l) + GetSize(r);
    sum = now + GetSum(l) + GetSum(r);
  }
  void Update(const T &val) {
    now = val; Update();
  }
  void Push() {
    Update(now + lz);
    if (flip)
      swap(l, r);
    for (auto c : {l, r}) if (c)
        c->flip ^= flip, c->lz += lz;
    lz = 0; flip = false;
  }
};
Node *rt;
Node *Splay(Node *x, Node *g = nullptr) {
  for (g || (rt = x); x->p != g; x->Rotate()) {
    if (!x->p->IsRoot()) x->p->p->Push();
    x->p->Push(); x->Push();
    if (x->p->p != g)
      (x->IsLeft() ^ x->p->IsLeft() ? x :
      x->p)->Rotate();
  }
  x->Push(); return x;
}
Node *Kth(int k) {
  for (auto x = rt;; x = x->r) {
    for (; x->Push(), x->l && x->l->sz > k; x = x->l);
    if (x->l) k -= x->l->sz;
    if (!k--) return Splay(x);
  }
}
Node *Gather(int s, int e) {
```

```
  auto t = Kth(e + 1);
  return Splay(t, Kth(s - 1))->l;
}
Node *Flip(int s, int e) {
  auto x = Gather(s, e);
  x->flip ^= 1;
  return x;
}
Node *Shift(int s, int e, int k) {
  if (k >= 0) { // shift to right
    k %= e - s + 1;
    if (k)
      Flip(s, e), Flip(s, s + k - 1), Flip(s + k, e);
  } else { // shift to left
    k = -k;
    k %= e - s + 1;
    if (k)
      Flip(s, e), Flip(s, e - k), Flip(e - k + 1, e);
  }
  return Gather(s, e);
}
int Idx(Node *x) { return x->l->sz; }
//////////// Link Cut Tree Start ////////////
Node *Splay(Node *x) {
  for (; !x->IsRoot(); x->Rotate()) {
    if (!x->p->IsRoot()) x->p->p->Push();
    x->p->Push(); x->Push();
    if (!x->p->IsRoot())
      (x->IsLeft() ^ x->p->IsLeft() ? x :
      x->p)->Rotate();
  }
  x->Push();
  return x;
}
void Access(Node *x) {
  Splay(x); x->r = nullptr;
  x->Update();
  for (auto y = x; x->p; Splay(x))
    y = x->p, Splay(y), y->r = x, y->Update();
}
int GetDepth(Node *x) {
  Access(x); x->Push();
  return GetSize(x->l);
}
Node *GetRoot(Node *x) {
  Access(x);
  for (x->Push(); x->l; x->Push())
    x = x->l;
  return Splay(x);
}
Node *GetPar(Node *x) {
  Access(x); x->Push();
  if (!x->l) return nullptr;
  x = x->l;
  for (x->Push(); x->r; x->Push())
    x = x->r;
  return Splay(x);
}
void Link(Node *p, Node *c) {
  Access(c); Access(p);
  c->l = p; p->p = c;
  c->Update();
}
void Cut(Node *c) {
  Access(c);
  c->l->p = nullptr;
  c->l = nullptr;
  c->Update();
}
Node *GetLCA(Node *x, Node *y) {
  Access(x); Access(y); Splay(x);
  return x->p ? x->p : x;
}
Node *Ancestor(Node *x, int k) {
  k = GetDepth(x) - k;
  assert(k >= 0);
  for (;; x->Push()) {
    int s = GetSize(x->l);
    if (s == k) return Access(x), x;
    if (s < k) k -= s + 1, x = x->r;
    else x = x->l;
  }
}
void MakeRoot(Node *x) {
  Access(x); Splay(x);
  x->flip ^= 1; x->Push();
}
bool IsConnect(Node *x, Node *y) { return GetRoot(x) ==
GetRoot(y); }
void PathUpdate(Node *x, Node *y, T val) {
  Node *root = GetRoot(x); // original root
  MakeRoot(x); Access(y); Splay(x);
  x->lz += val; x->Push();
  MakeRoot(root); // Revert
  Node *lca = GetLCA(x, y);
  Access(lca); Splay(lca);
  lca->Push(); lca->Update(lca->now - val);
}
T VertexQuery(Node *x, Node *y) {
  Node *l = GetLCA(x, y); T ret = l->now;
  Access(x); Splay(l);
  if (l->r) ret = ret + l->r->sum;
  Access(y); Splay(l);
  if (l->r) ret = ret + l->r->sum;
  return ret;
}
Node *GetQueryResultNode(Node *u, Node *v) {
  if (!IsConnect(u, v)) return 0;
  MakeRoot(u); Access(v);
  auto ret = v->l;
  while (ret->mx != ret->now) {
    if (ret->l && ret->mx == ret->l->mx)
      ret = ret->l;
    else ret = ret->r;
  }
  Access(ret); return ret;
}
```

4 Graph

4.1 Bellman Ford

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

```
auto bellman = [&](int s) -> bool {
  fill(all(d), INF); d[s] = 0; bool chk = 0;
  for (int i = 0; i < n; i++) { chk = 0;
    for (int u = 1; u <= n; u++) {
      if (d[u] == INF) continue;
      for (auto [w, v] : adj[u]) if (d[v] > d[u] + w) {
        d[v] = d[u] + w; chk = 1; if (i == n - 1)
        return 0;
      }
    }
    if (!chk) break;
  }
  return 1;
};
```

4.2 SPFA (SLF Optimized)

Usage: SSSP with negative weights. Returns false if negative cycle detected.
Time Complexity: Avg $O(E)$, Worst $O(VE)$

```
auto spfa = [&](int s) -> bool {
  vector<int> c(n+1), inq(n+1); fill(all(d), INF);
  deque<int> q; q.push_back(s); d[s] = 0; inq[s] = 1;
  while (!q.empty()) {
    int u = q.front(); q.pop_front(); inq[u] = 0;
    for (auto [w, v] : adj[u]) if (d[v] > d[u] + w) {
      d[v] = d[u] + w; if (inq[v]) continue;
      if (sz(q) && d[v] < d[q.front()])
      q.push_front(v);
```

```
    else q.push_back(v);
    inq[v] = 1; if (++c[v] >= n) return 0;
   }
  }
  return 1;
};
```

4.3 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
```

4.4 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 n, idx;
  vector<vector<int>> adj, graph;
  HLD (int n_) : n(n_), dep(n+1), par(n+1), sz(n+1),
  in(n+1), out(n+1), top(n+1), adj(n+1), graph(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);
  }
};
```

4.5 Centroid Decomposition

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

```
struct CentroidTree {
  vector<vector<int>> adj, c_adj; // adj: 원본트리 /
  c_adj: 센트로이드트리
  vector<int> sz, par, vis; int N;
  CentroidTree(int n) : N(n), adj(n+1), c_adj(n+1),
  sz(n+1), par(n+1), vis(n+1) {}
  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_cent(int curr, int prev, int to_sz) {
    for (int next : adj[curr])
      if (next != prev && !vis[next] && sz[next] >
      to_sz / 2)
        return get_cent(next, curr, to_sz);
    return curr;
  }
  void solve(int u) { /* 현재 센트로이드를 포함하는 모
  든 경로를 계산하는 로직 */ }
  int build(int curr, int p = -1) {
    int cent = get_cent(curr, -1, get_sz(curr, -1));
    solve(cent);
    vis[cent] = 1; par[cent] = p;
    for (int next : adj[cent]) {
      if (!vis[next]) {
        int child = build(next, cent);
        c_adj[cent].push_back(child); // 센트로이드 계
        층 연결
      }
    }
```

```
    return cent;
  }
};
```

4.6 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;
  vector<bool> visA, visB, fA, fB; // vertex i can be
  excluded from some max matching
  int ns, ms;
  BiMatch(int n, int m) : ns(n), ms(m), graph(n+1),
  grev(m+1), mA(n+1), mB(m+1), dist(n+1), work(n+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], 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;
  }
  void chkEss() {
    fA.assign(ns+1, 0); fB.assign(ms+1, 0);
    visA.assign(ns+1, 0); visB.assign(ms+1, 0);
    queue<int> q;
    for (int i = 1; i <= ns; i++) if (!mA[i]) fA[i] =
    visA[i] = 1, q.push(i);
    while (!q.empty()) {
      int u = q.front(); q.pop();
      for (int v : graph[u]) if (!visB[v]) {
        visB[v] = 1; int r = mB[v];
        if (r && !fA[r]) {
          fA[r] = visA[r] = 1; q.push(r);
        }
      }
    }
    for (int i = 1; i <= ms; i++) if (!mB[i]) fB[i] =
    1, q.push(i);
    while (!q.empty()) {
      int v = q.front(); q.pop();
      for (int u : grev[v]) {
        int r = mA[u];
        if (r && !fB[r]) {
          fB[r] = 1; q.push(r);
        }
      }
    }
  }
  pair<vector<int>,vector<int>> vertex() { // find
  minimum vertex cover
    chkEss();
    vector<int> va, vb;
    for (int i = 1; i <= ns; i++) if (!visA[i])
    va.push_back(i);
    for (int i = 1; i <= ms; i++) if (visB[i])
    vb.push_back(i);
    return { va, vb };
  }
};
/* struct BiMatch {
  vector<vector<int>> graph;
  vector<int> mA, mB, vis;
  int ns, ms;
  BiMatch(int n, int m) : ns(n), ms(m), graph(n+1),
  mA(n+1), mB(m+1), vis(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;
  }
}; */
```

4.7 Dinic

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

```
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(n+1), level(n+1),
  work(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;
  }
};
```

4.8 MCMF

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

```
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(n+1), dist(n+1),
  parent(n+1), edge(n+1), vis(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 };
  }
};
```

4.9 Circulation

Usage: Flow with lower and upper bounds.

```
const ll INF = 1e18;
struct Dinic {}; // or MCMF
struct Circulation {
  vector<ll> demand, low;
  vector<pair<int,int>> edge;
  int n, S, T; 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, res = 0;
    for (int i = 1; i <= n; i++) sum += demand[i];
    if (sum != 0) return false;
    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];
  }
};
```

4.10 SCC

Usage: Strongly Connected Components.
Time Complexity: $O(V + E)$

```
struct SCC {
  int n, cnt, t;
  vector<vector<int>> adj;
  vector<int> dfn, low, id;
  vector<bool> ins; stack<int> st;
  SCC(int n) : n(n), adj(n), dfn(n, -1), low(n, -1),
  id(n, -1), ins(n), cnt(0), t(0) {}
  void add(int u, int v) { adj[u].push_back(v); }
  void dfs(int u) {
    dfn[u] = low[u] = ++t;
    st.push(u); ins[u] = true;
```

```
    for (int v : adj[u]) {
      if (dfn[v] == -1) {
        dfs(v); low[u] = min(low[u], low[v]);
      } else if (ins[v]) {
        low[u] = min(low[u], dfn[v]);
      }
    }
    if (low[u] == dfn[u]) {
      while (true) {
        int v = st.top(); st.pop();
        ins[v] = false; id[v] = cnt;
        if (u == v) break;
      }
      cnt++;
    }
  }
  void build() {
    for (int i = 0; i < n; i++)
      if (dfn[i] == -1) dfs(i);
  }
};
```

4.11 2-sat

Usage: Solves 2-SAT in $O(V + E)$; x_i is true if $SCC(x_i) < SCC(\neg x_i)$.

Time Complexity: $O(V + E)$

```
struct TwoSat {
  int n; SCC scc;
  vector<bool> res;
  TwoSat(int n) : n(n), scc(2 * n + 2), res(n + 1) {}
  // (x_a == is_a) OR (x_b == is_b) 추가
  void add(int a, bool is_a, int b, bool is_b) {
    int u = a << 1 | !is_a, v = b << 1 | !is_b;
    scc.add(u ^ 1, v);
    scc.add(v ^ 1, u);
  }
  bool satisfiable() {
    scc.build();
    for (int i = 1; i <= n; i++) {
      if (scc.id[i << 1] == scc.id[i << 1 | 1]) return
      false;
      res[i] = scc.id[i << 1] < scc.id[i << 1 | 1];
    }
    return true;
  }
  bool get(int i) { return res[i]; }
};
```

4.12 BCC

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

```
// 1-based, 다른 거 호출하기 전에 tarjan 먼저 호출
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);
}
```

4.13 Find 3 or 4 cycle

Usage: Finds all C_3 and C_4 cycles using degree ordering.
Time Complexity: $O(M\sqrt{M})$

```
vector<tuple<int,int,int>> Find3Cycle(int n, const
vector<pair<int,int>> &edges){ // N+MsqrtN
  int m = edges.size();
  vector<int> deg(n), pos(n), ord; ord.reserve(n);
  vector<vector<int>> gph(n), que(m+1), vec(n);
  vector<vector<tuple<int,int,int>>> tri(n);
  vector<tuple<int,int,int>> res;
  for(auto [u,v] : edges) deg[u]++, deg[v]++;
  for(int i=0; i<n; i++) que[deg[i]].push_back(i);
  for(int i=m; i>=0; i--) ord.insert(ord.end(),
  que[i].begin(), que[i].end());
  for(int i=0; i<n; i++) pos[ord[i]] = i;
  for(auto [u,v] : edges)
  gph[pos[u]].push_back(pos[v]),
  gph[pos[v]].push_back(pos[u]);
  for(int i=0; i<n; i++){
    for(auto j : gph[i]){
      if(i > j) continue;
      for(int x=0, y=0; x<vec[i].size() &&
      y<vec[j].size(); ){
```

```
        if(vec[i][x] == vec[j][y])
        res.emplace_back(ord[i], ord[j], ord[vec[i]
        [x]]), x++, y++;
        else if(vec[i][x] < vec[j][y]) x++; else y++;
      }
      vec[j].push_back(i);
    }
  }
  for(auto &[u,v,w] : res){
    if(pos[u] < pos[v]) swap(u, v);
    if(pos[u] < pos[w]) swap(u, w);
    if(pos[v] < pos[w]) swap(v, w);
    tri[u].emplace_back(u, v, w);
  }
  res.clear();
  for(int i=n-1; i>=0; i--) res.insert(res.end(),
  tri[ord[i]].begin(), tri[ord[i]].end());
  return res;
}
bitset<500> B[500]; // N3/w
long long Count3Cycle(int n, const
vector<pair<int,int>> &edges){
  long long res = 0;
  for(int i=0; i<n; i++) B[i].reset();
  for(auto [u,v] : edges) B[u].set(v), B[v].set(u);
  for(int i=0; i<n; i++) for(int j=i+1; j<n; j++)
  if(B[i].test(j)) res += (B[i] & B[j]).count();
  return res / 3;
}
// O(n + m * sqrt(m) + th) for graphs without loops or
multiedges
void Find4Cycle(int n, const vector<array<int, 2>>
&edge, auto process, int th = 1){
  int m = (int)edge.size();
  vector<int> deg(n), order, pos(n);
  vector<vector<int>> appear(m+1), adj(n), found(n);
  for(auto [u, v]: edge) ++deg[u], ++deg[v];
  for(auto u=0; u<n; u++) appear[deg[u]].push_back(u);
  for(auto d=m; d>=0; d--) order.insert(order.end(),
  appear[d].begin(), appear[d].end());
  for(auto i=0; i<n; i++) pos[order[i]] = i;
  for(auto i=0; i<m; i++){
    int u = pos[edge[i][0]], v = pos[edge[i][1]];
    adj[u].push_back(v), adj[v].push_back(u);
  }
  T res = 0; vector<int> cnt(n);
  for(auto u=0; u<n; u++){
    for(auto v: adj[u]) if(u < v) for(auto w: adj[v])
    if(u < w) cnt[w] = 0;
    for(auto v: adj[u]) if(u < v) for(auto w: adj[v])
    if(u < w) res += cnt[w] ++;
  }
  for(auto u=0; u<n; u++){
    for(auto v: adj[u]) if(u < v) for(auto w: adj[v])
    if(u < w) found[w].clear();
    for(auto v: adj[u]) if(u < v) for(auto w: adj[v])
    if(u < w) {
      for(auto x: found[w]){
        if(!th--) return;
        process(order[u], order[v], order[w],
        order[x]);
      }
      found[w].push_back(v);
    }
  }
}
```

4.14 Push Relabel

Usage: Max flow via preflow-push and labeling
Time Complexity: $O(V^3)$

```
const ll INF = 1e18;
struct HLPP {
  struct Edge {
    int to; ll cap; int rev;
  };
  vector<vector<Edge>> graph;
  vector<ll> ex;
  vector<int> level, work;
  vector<vector<int>> B;
  int n, high = 0, cnt = 0;
  HLPP(int n) : n(n+1), graph(n+1), level(n+1),
  work(n+1), ex(n+1), B(n+1) {}
  void add(int u, int v, ll cap) {
    graph[u].push_back({ v, cap, sz(graph[v]) });
    graph[v].push_back({ u, cap, sz(graph[u])-1 });
  }
  void push(int u) {
    if (level[u] >= n) return;
    B[level[u]].push_back(u);
    high = max(high, level[u]);
  }
  void relabel(int t) {
    cnt = 0;
    for (auto& b : B) b.clear();
    fill(all(level), n);
    queue<int> q; q.push(t);
    level[t] = 0;
    while (!q.empty()) {
      int u = q.front(); q.pop();
      int h = level[u] + 1;
      for (auto& e : graph[u]) {
        if (graph[e.to][e.rev].cap > 0 && h <
        level[e.to]) {
          level[e.to] = h; q.push(e.to);
          if (ex[e.to] > 0) push(e.to);
        }
      }
    }
  }
  void discharge(int u) {
    auto& excess = ex[u];
    int h = n * 2, sz = sz(graph[u]);
    for (int& i = work[u], m = sz; m--; i = (i-1 + sz)
    % sz) {
      auto& e = graph[u][i];
      if (!e.cap) continue;
      if (level[u] != level[e.to] + 1) {
        h = min(h, level[e.to] + 1);
        continue;
      }
      auto f = min(e.cap, excess);
      e.cap -= f;
      excess -= f;
      if (!ex[e.to]) push(e.to);
      ex[e.to] += f;
      graph[e.to][e.rev].cap += f;
      if (!excess) return;
    }
    cnt++; level[u] = h;
    if (level[u] < n && ex[u] > 0) push(u);
  }
  ll flow(int s, int t) {
    relabel(t);
    ex[s] = INF; ex[t] = -INF;
    push(s);
    for (; ~high; high--) {
      while (!B[high].empty()) {
        int u = B[high].back();
        B[high].pop_back();
        if (level[u] == high) discharge(u);
        if (cnt > n/8) relabel(t);
      }
    }
    return ex[t] + INF;
  }
};
```

4.15 General Matching

Usage: Maximum Unweighted Matching.
Time Complexity: $O(N^3)$

```
struct GeneralMatch {
  vector<vector<int>> graph;
  vector<int> vis, parent, orig, matched, aux;
  queue<int> q; int n, t = 0;
  GeneralMatch(int n) : n(n) {
    auto init = [&](auto&... vecs) { (vecs.resize(n+1),
    ...); };
    init(graph, vis, parent, orig, matched, aux);
  }
  void add(int a, int b) { graph[a].push_back(b);
  graph[b].push_back(a); }
  void augment(int u, int v) {
    while (v) {
      int pv = parent[v], nv = matched[pv];
      matched[v] = pv; matched[pv] = v;
      v = nv;
    }
  }
  int lca(int v, int w) {
    ++t;
    while (1) {
      if (v) {
        if (aux[v] == t) return v;
        aux[v] = t; v = orig[parent[matched[v]]];
      }
      swap(v, w);
    }
  }
  void blossom(int v, int w, int a) {
    while (orig[v] != a) {
      parent[v] = w; w = matched[v];
      if (vis[w] == 1) { q.push(w), vis[w] = 0; }
      orig[v] = orig[w] = a; v = parent[w];
    }
  }
  bool bfs(int u, int ban = 0) {
    fill(all(vis), -1); iota(all(orig), 0);
    if (ban) vis[ban] = 2;
    q = queue<int>(); q.push(u); vis[u] = 0;
    while (!q.empty()) {
      int v = q.front(); q.pop();
      for (auto w : graph[v]) {
        if (vis[w] == -1) {
          parent[w] = v; vis[w] = 1;
          if (!matched[w]) { augment(u, w); return
          true; }
          vis[matched[w]] = 0; q.push(matched[w]);
        }
        else if (vis[w] == 0 && orig[v] != orig[w]) {
          int a = lca(orig[v], orig[w]);
          blossom(w, v, a); blossom(v, w, a);
        }
      }
    }
    return false;
  }
  int match() {
    int ans = 0;
    for(int i = 1; i <= n; i++) {
      if(!matched[i] && bfs(i)) ans++;
    }
    return ans;
  }
  bool chk(int u) { // find max matching except u
    if (!matched[u]) return true;
    auto backup = matched;
    int v = matched[u];
    matched[u] = matched[v] = 0;
    bool res = bfs(v, u);
    matched = backup;
    return res;
  }
};
```

4.16 Weighed General Matching

Usage: Maximum Weighted Matching.
Time Complexity: $O(N^3)$

```
const ll INF = 1e18;
struct WeightedGeneralMatch {
  struct Edge { int u = 0, v = 0; ll w = 0; };
  int n, nx;
  vector<vector<Edge>> adj; vector<ll> label;
  vector<int> matched, slack, root, parent, state, vis;
  vector<vector<int>> node, via;
  queue<int> q; int time = 0;
  WeightedGeneralMatch(int n) : n(n), nx(n) {
    int size = 2*n+1;
    auto init = [&](auto&... vecs) {
    (vecs.resize(size), ...); };
    init(label, matched, slack, root, parent, state,
    vis, node);
    adj.resize(size, vector<Edge>(size));
    via.resize(size, vector<int>(n+1));
    for (int u = 1; u <= n; u++)
      for (int v = 1; v <= n; v++)
        adj[u][v] = { u, v, 0 };
  }
  void add(int u, int v, ll w) { adj[u][v].w = adj[v]
  [u].w = w; }
  ll calc(Edge e) { return label[e.u] + label[e.v] -
  adj[e.u][e.v].w * 2; }
  void minSlack(int u, int x) {
    if (!slack[x] || calc(adj[u][x]) <
    calc(adj[slack[x]][x]))
      slack[x] = u;
  }
  void updSlack(int x) {
    slack[x] = 0;
    for (int u = 1; u <= n; u++) {
      if (adj[u][x].w > 0 && root[u] != x &&
      state[root[u]] == 0) {
        minSlack(u, x);
      }
    }
  }
  void updRoot(int x, int b) {
    root[x] = b;
    if (x > n) for (auto i : node[x]) updRoot(i, b);
  }
  void enqueue(int x) {
    if (x <= n) q.push(x);
    else for (auto i : node[x]) enqueue(i);
  }
  int getIdx(int b, int xr) {
    int idx = find(all(node[b]), xr) - node[b].begin();
    if (idx % 2 == 1) {
      reverse(node[b].begin() + 1, node[b].end());
      return sz(node[b]) - idx;
    }
    return idx;
  }
  void rematch(int u, int v) {
    matched[u] = adj[u][v].v;
    if (u <= n) return;
    Edge e = adj[u][v];
    int xr = via[u][e.u], pr = getIdx(u, xr);
    for (int i = 0; i < pr; i++)
      rematch(node[u][i], node[u][i ^ 1]);
    rematch(xr, v);
    rotate(node[u].begin(), node[u].begin() + pr,
    node[u].end());
  }
  void augment(int u, int v) {
    while (true) {
      int xnv = root[matched[u]];
      rematch(u, v);
      if (!xnv) return;
      rematch(xnv, root[parent[xnv]]);
      u = root[parent[xnv]]; v = xnv;
    }
  }
  int findLCA(int u, int v) {
```

```
    time++;
    while (u || v) {
      if (u == 0) { swap(u, v); continue; }
      if (vis[u] == time) return u;
      vis[u] = time; u = root[matched[u]];
      if (u) u = root[parent[u]];
      swap(u, v);
    }
    return 0;
  }
  void addBlossom(int u, int lca, int v) {
    int b = n+1;
    while (b <= nx && root[b]) b++;
    if (b > nx) nx++;
    label[b] = 0, state[b] = 0;
    matched[b] = matched[lca];
    node[b].clear(); node[b].push_back(lca);
    for (int x = u, y; x != lca; x = root[parent[y]]) {
      node[b].push_back(x), node[b].push_back(y =
      root[matched[x]]);
      enqueue(y);
    }
    reverse(node[b].begin() + 1, node[b].end());
    for (int x = v, y; x != lca; x = root[parent[y]]) {
      node[b].push_back(x), node[b].push_back(y =
      root[matched[x]]);
      enqueue(y);
    }
    updRoot(b, b);
    for (int x = 1; x <= nx; x++)
      adj[b][x].w = adj[x][b].w = 0;
    for (int x = 1; x <= n; x++) via[b][x] = 0;
    for (int xs : node[b]) {
      for (int x = 1; x <= nx; x++) {
        if (adj[b][x].w == 0 || calc(adj[xs][x]) <
        calc(adj[b][x])) {
          adj[b][x] = adj[xs][x];
          adj[x][b] = adj[x][xs];
        }
      }
      for (int x = 1; x <= n; x++)
        if (via[xs][x])
          via[b][x] = xs;
    }
    updSlack(b);
  }
  void expandBlossom(int b) {
    for (auto i : node[b]) updRoot(i, i);
    int xr = via[b][adj[b][parent[b]].u], pr =
    getIdx(b, xr);
    for (int i = 0; i < pr; i+=2) {
      int xs = node[b][i], xns = node[b][i + 1];
      parent[xs] = adj[xns][xs].u;
      state[xs] = 1; state[xns] = slack[xs] = 0;
      updSlack(xns); enqueue(xns);
    }
    state[xr] = 1; parent[xr] = parent[b];
    for (int i = pr+1; i < sz(node[b]); i++) {
      int xs = node[b][i];
      state[xs] = -1; updSlack(xs);
    }
    root[b] = 0;
  }
  bool inspect(Edge e) {
    int u = root[e.u], v = root[e.v];
    if (state[v] == -1) {
      parent[v] = e.u; state[v] = 1;
      int nu = root[matched[v]];
      slack[v] = slack[nu] = 0;
      state[nu] = 0; enqueue(nu);
    }
    else if (state[v] == 0) {
      int lca = findLCA(u, v);
      if (!lca) {
        augment(u, v); augment(v, u);
        return true;
      }
      else addBlossom(u, lca, v);
    }
    return false;
  }
  bool matching() {
    fill(state.begin(), state.begin() + nx + 1, -1);
    fill(slack.begin(), slack.begin() + nx + 1, 0);
    q = queue<int>();
    for (int x = 1; x <= nx; x++) {
      if (root[x] == x && !matched[x]) {
        parent[x] = 0; state[x] = 0;
        enqueue(x);
      }
    }
    if (q.empty()) return false;
    while (true) {
      while (!q.empty()) {
        int u = q.front(); q.pop();
        if (state[root[u]] == 1) continue;
        for (int v = 1; v <= n; v++) {
          if (adj[u][v].w > 0 && root[u] != root[v]) {
            if (calc(adj[u][v]) == 0) {
              if (inspect(adj[u][v])) return true;
            }
            else minSlack(u, root[v]);
          }
        }
      }
      ll d = INF;
      for (int b = n+1; b <= nx; b++) {
        if (root[b] == b && state[b] == 1) {
          d = min(d, label[b] / 2);
        }
      }
      for (int x = 1; x <= nx; x++) {
        if (root[x] == x && slack[x]) {
          if (state[x] == -1) d = min(d,
          calc(adj[slack[x]][x]));
          else if (state[x] == 0) d = min(d,
          calc(adj[slack[x]][x]) / 2);
        }
      }
      for (int u = 1; u <= n; u++) {
        if (state[root[u]] == 0) {
          if (label[u] <= d) return false;
          label[u] -= d;
        }
        else if (state[root[u]] == 1) label[u] += d;
      }
      for (int b = n + 1; b <= nx; b++) {
        if (root[b] == b) {
          if (state[root[b]] == 0) label[b] += d * 2;
          else if (state[root[b]] == 1) label[b] -= d *
          2;
        }
      }
      q = queue<int>();
      for (int x = 1; x <= nx; x++) {
        if (root[x] == x && slack[x] && root[slack[x]]
        != x && calc(adj[slack[x]][x]) == 0) {
          if (inspect(adj[slack[x]][x])) return true;
        }
      }
      for (int b = n + 1; b <= nx; b++) {
        if (root[b] == b && state[b] == 1 && label[b]
        == 0) expandBlossom(b);
      }
    }
    return false;
  }
  pair<ll,int> match() {
    fill(matched.begin(), matched.begin() + n+1, 0);
    nx = n;
    ll cost = 0; int res = 0;
    for (int u = 0; u <= n; u++) {
      root[u] = u;
```

```
      node[u].clear();
    }
    ll maxw = 0;
    for (int u = 1; u <= n; u++) {
      for (int v = 1; v <= n; v++) {
        via[u][v] = (u == v ? u : 0);
        maxw = max(maxw, adj[u][v].w);
      }
    }
    for (int u = 1; u <= n; u++) label[u] = maxw;
    while (matching()) res++;
    for (int u = 1; u <= n; u++) {
      if (matched[u] && matched[u] < u) {
        cost += adj[u][matched[u]].w;
      }
    }
    return { cost, res };
  }
};
```

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)
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).
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;
```

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];
// l부터 r까지 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: $O(N \log N)$

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

```cpp
    }
    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: $O(T \log X)$

```cpp
/* 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: $O(N2^N)$

```cpp
// 1. Subset Sum: f[mask] = sum of a[sub] where sub &
mask == sub
for (int i = 0; i < n; ++i)
    for (int mask = 0; mask < (1 << n); ++mask)
        if (mask & (1 << i)) f[mask] += f[mask ^ (1 <<
        i)];

// 2. Superset Sum: f[mask] = sum of a[sup] where sup &
mask == mask
for (int i = 0; i < n; ++i)
    for (int mask = (1 << n) - 1; mask >= 0; --mask)
        if (!(mask & (1 << i))) f[mask] += f[mask ^ (1
        << i)];
```

5.7 Berlekamp Massey

Usage: Linear recurrence N-th term. Sparse matrix determinant.
Time Complexity: N-th term: $O(K^2 \log N)$ / Det: $O(N(N+E))$

```cpp
mt19937_64
rng(chrono::high_resolution_clock::now().time_since_epoch().count());
int randint(int lb, int ub) { return
uniform_int_distribution<int>(lb, ub)(rng); }
const int mod = 998244353;
ll ipow(ll x, ll p) {
  ll ret = 1, piv = x;
  while (p) {
    if (p & 1) ret = ret * piv % mod;
    piv = piv * piv % mod; p >>= 1;
  }
  return ret;
}
vector<int> berlekamp_massey(vector<int> x) {
  vector<int> ls, cur; int lf, ld;
  for (int i = 0; i < sz(x); i++) {
    ll t = 0;
    for (int j = 0; j < sz(cur); j++) t = (t +
    1ll*x[i-j-1]*cur[j]) % mod;
    if ((t - x[i]) % mod == 0) continue;
    if (cur.empty()) {
      lf = i; ld = (t-x[i]) % mod;
      cur.resize(i+1); continue;
    }
    ll k = -(x[i]-t) * ipow(ld, mod-2) % mod;
    vector<int> c(i-lf-1); c.push_back(k);
    for (auto& j : ls) c.push_back(-j*k % mod);
    if (sz(c) < sz(cur)) c.resize(sz(cur));
    for (int j = 0; j < sz(cur); j++) c[j] =
    (c[j]+cur[j]) % mod;
    if (i-lf+sz(ls) >= sz(cur)) {
      tie(ls, lf, ld) = make_tuple(cur, i, (t-x[i]) %
      mod);
    } cur = c;
  }
  for (auto& i : cur) i = (i % mod + mod) % mod;
  return cur;
}
int get_nth(vector<int> rec, vector<int> dp, ll n) {
  int m = sz(rec); vector<int> s(m), t(m); s[0] = 1;
  if(m != 1) t[1] = 1; else t[0] = rec[0];
  auto mul = [&rec](vector<int> v, vector<int> w) {
    int m = sz(v); vector<int> t(2*m);
    for (int j = 0; j < m; j++) {
      for (int k = 0; k < m; k++) {
        t[j+k] += 1ll*v[j]*w[k] % mod;
        if (t[j+k] >= mod) t[j+k] -= mod;
      }
    }
    for (int j = 2*m-1; j >= m; j--) {
      for (int k = 1; k <= m; k++) {
        t[j-k] += 1ll*t[j]*rec[k-1] % mod;
        if (t[j-k] >= mod) t[j-k] -= mod;
      }
    }
    t.resize(m); return t;
  };
  while (n) {
```

```
    if (n & 1) s = mul(s, t);
    t = mul(t, t); n >>= 1;
  }
  ll ret = 0;
  for (int i = 0; i < m; i++) ret += 1ll*s[i]*dp[i] %
  mod;
  return ret % mod;
}
int guess_nth_term(vector<int> x, ll n) {
  if (n < sz(x)) return x[n];
  vector<int> v = berlekamp_massey(x);
  if (v.empty()) return 0;
  return get_nth(v, x, n);
}
struct elem { int x, y, v; }; // A_(x, y) <- v,
0-based. no duplicate
vector<int> get_min_poly(int n, vector<elem> M) {
  // smallest poly P such that A^i = sum_{j < i} {A^j
  \times P_j}
  vector<int> rnd1, rnd2;
  for (int i = 0; i < n; i++) {
    rnd1.push_back(randint(1, mod - 1));
    rnd2.push_back(randint(1, mod - 1));
  }
  vector<int> gobs;
  for (int i = 0; i < 2*n+2; i++) {
    int tmp = 0;
    for (int j = 0; j < n; j++) {
      tmp += 1ll * rnd2[j] * rnd1[j] % mod;
      if (tmp >= mod) tmp -= mod;
    }
    gobs.push_back(tmp); vector<int> nxt(n);
    for (auto& i : M) {
      nxt[i.x] += 1ll * i.v * rnd1[i.y] % mod;
      if(nxt[i.x] >= mod) nxt[i.x] -= mod;
    }
    rnd1 = nxt;
  }
  auto sol = berlekamp_massey(gobs);
  reverse(all(sol)); return sol;
}
ll det(int n, vector<elem> M) {
  vector<int> rnd;
  for (int i = 0; i < n; i++) rnd.push_back(randint(1,
  mod-1));
  for (auto& i : M) i.v = 1ll * i.v * rnd[i.y] % mod;
  auto sol = get_min_poly(n, M)[0]; if (n%2 == 0) sol =
  mod-sol;
  for (auto& i : rnd) sol = 1ll * sol * ipow(i, mod-2)
  % mod;
  return sol;
}
```

6 Geometry

6.1 Geometry Template

Usage: Basic point, line, and circle operations.

```
const double EPS = 1e-9;
template<typename T>
struct Point {
  T x, y;
  bool operator<(const Point& p) const { return x==p.x
  ? y<p.y : x<p.x; }
  bool operator==(const Point& p) const { return x==p.x
  && y==p.y; }
  Point operator+(const Point& p) const { return
  {x+p.x, y+p.y}; }
  Point operator-(const Point& p) const { return
  {x-p.x, y-p.y}; }
  Point operator*(T n) const { return {x*n, y*n}; }
  Point operator/(T n) const { return {x/n, y/n}; }
  T operator*(const Point& p) const { return x*p.x +
  y*p.y; }
  T operator/(const Point& p) const { return x*p.y -
  y*p.x; }
  /* 3D dot, cross
  T operator*(const Point& p) const { return x*p.x +
  y*p.y + z*p.z; }
  Point operator/(const Point& p) const {
    return {y*p.z - z*p.y, z*p.x - x*p.z, x*p.y -
    y*p.x};
  } */
  T dist2() const { return x*x + y*y; }
  Point<double> d() const { return {(double)x,
  (double)y}; }
  Point rot90() const { return {-y, x}; }
};
using P = Point<ll>;
using Pd = Point<double>;
int ccw(P a, P b, P c) {
  ll res = (b - a) / (c - a);
  return (res > 0) - (res < 0);
}
bool isInter(P a, P b, P c, P d) {
  int ab = ccw(a, b, c) * ccw(a, b, d), cd = ccw(c, d,
  a) * ccw(c, d, b);
  if (ab == 0 && cd == 0) {
    if (b < a) swap(a, b); if (d < c) swap(c, d);
    return !(b < c || d < a);
  }
  return ab <= 0 && cd <= 0;
}
```

6.2 Convex Hull

Usage: Finds the convex hull using Monotone Chain. Returns vertices in CCW order.
Time Complexity: $O(N \log N)$

```
vector<P> ConvexHull(vector<P> ps) { // Monotone chain
  if (sz(ps) <= 2) return ps;
  sort(all(ps)); vector<P> v(sz(ps)+2);
  int s = 0, t = 0;
  for (int i = 2; i--; s = --t, reverse(all(ps))) {
    for (P p : ps) { // include boundary : ccw < 0
      while (t >= s+2 && ccw(v[t-2], v[t-1], p) <= 0)
      t--;
      v[t++] = p;
    }
  }
  v.resize(t - (t > 1)); return v;
}
```

6.3 Rotating Calipers

Usage: Calculates the diameter (farthest pair of points) of a convex hull(CCW order).
Time Complexity: $O(N)$

```
// ccw 정렬된 convex에서 가장 먼 두 점
pair<P,P> Calipers(const vector<P>& v) {
  int n = sz(v); if (n < 2) return {v[0], v[0]};
  ll mx = 0; P a = v[0], b = v[1];
  for (int i = 0, j = 1; i < n; i++) {
    P t = v[(i+1)%n] - v[i];
    while ((t / (v[(j+1)%n] - v[j])) > 0) j = (j+1)%n;
    ll now = (v[i] - v[j]).dist2();
    if (now > mx) mx = now, a = v[i], b = v[j];
  }
  return { a, b };
}
```

6.4 Point in Convex Polygon

Usage: Checks if a point is inside or on the boundary of a convex polygon (CCW sorted).
Time Complexity: $O(\log N)$

```
// CCW 정렬된 다각형 내부 판별, O(log N)
bool InConvPoly(const vector<P>& v, P p) {
  int n = sz(v); if (n<3) return false;
  // ccw <= 0 || ccw >= 0: exclude boundary
```

```cpp
  if (ccw(v[0], v[1], p) < 0 || ccw(v[0], v.back(), p)
  > 0) return false;
  int l = 1, r = n - 1;
  while (l + 1 < r) {
    int mid = (l + r) / 2;
    if (ccw(v[0], v[mid], p) >= 0) l = mid;
    else r = mid;
  }
  return ccw(v[l], v[l + 1], p) >= 0; // > 0: exclude
  boundary
}
```

6.5 Point in Polygon

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

```cpp
// 다각형 내부 판별 (CCW/CW 순서 무관)
bool InPoly(const vector<P>& v, P p) {
  int n = sz(v); bool chk = false;
  for (int i = 0; i < n; i++) {
    P a = v[i], b = v[(i + 1) % n];
    if ((b - a) / (p - a) == 0 && (a - p) * (b - p) <=
    0) return true; // false: exclude boundary
    if ((a.y > p.y) != (b.y > p.y)) {
      double ix = (double)(b.x-a.x) * (p.y-a.y) /
      (double)(b.y-a.y) + a.x;
      if (p.x < ix) chk = !chk;
    }
  }
  return chk;
}
```

6.6 Sort Points

Usage: Angular sort. 1. Relative to pivot (Convex Hull). 2. Relative to origin.

```cpp
// Sorts points by angle relative to the bottom-left
point (CCW).
void SortByAngle(vector<P>& v) {
  if (v.size() < 2) return;
  swap(v[0], *min_element(v.begin(), v.end(), [](const
  P& a, const P& b) {
    return a.y != b.y ? a.y < b.y : a.x < b.x;
  }));
  sort(v.begin() + 1, v.end(), [&](const P& a, const P&
  b) {
    ll cp = (a - v[0]) / (b - v[0]); if (cp != 0)
    return cp > 0;
    return (a - v[0]).dist2() < (b - v[0]).dist2();
  });
}
// Sorts points by angle around a given point 'cent' in
the range [0, 360).
void SortAroundPoint(vector<P>& v, P cent = {0, 0}) {
  auto half = [](const P& p) { return p.y > 0 || (p.y
  == 0 && p.x > 0); };
  sort(v.begin(), v.end(), [&](const P& a, const P& b)
  {
    P aa = a-cent, bb = b-cent;
    if (half(aa) != half(bb)) return half(aa) >
    half(bb);
    return (aa / bb) > 0;
  });
}
```

6.7 Linear Minkowski Sum

Usage: Minkowski Sum of Two convex(must be CCW order).
Time Complexity: $O(N + M)$

```cpp
// Minkowski Sum of Convex Polygons (CCW only), O(N +
M)
vector<P> Minkowski(vector<P> p, vector<P> q) {
  if (p.empty() || q.empty()) return {};
  rotate(p.begin(), min_element(all(p)), p.end());
  rotate(q.begin(), min_element(all(q)), q.end());
  p.push_back(p[0]); p.push_back(p[1]);
  q.push_back(q[0]); q.push_back(q[1]);
  vector<P> res; int i = 0, j = 0;
  while (i < p.size() - 2 || j < q.size() - 2) {
    res.push_back(p[i] + q[j]);
    ll cp = (p[i + 1] - p[i]) / (q[j + 1] - q[j]);
    if (cp >= 0 && i < p.size() - 2) i++;
    if (cp <= 0 && j < q.size() - 2) j++;
  }
  return res;
}
```

6.8 Polygon Area

Usage: Calculates 2 × Area of a polygon (Shoelace formula).
Time Complexity: $O(N)$

```cpp
// 다각형 넓이*2 (신발끈 공식)
ll area2(const vector<P>& v) {
  ll res = 0; int n = sz(v);
  for (int i = 0; i < n; i++)
    res += v[i] / v[(i+1)%n];
  return abs(res);
}
```

6.9 Smallest Enclosing Circle

Usage: Welzl's algorithm to find the Minimum Enclosing Circle. Returns center, radius.

Time Complexity: Expected $O(N)$

```cpp
Pd getC(Pd a, Pd b) { return (a + b) / 2.0; }
Pd getC(Pd a, Pd b, Pd c) {
  Pd u = b-a, v = c-a; double d = 2*(u/v);
  if (abs(d) < EPS) return {0, 0};
  return a - (v * u.dist2() - u * v.dist2()).rot90() /
  d;
}
/* // for 3D
Pd getC(Pd a, Pd b, Pd c) {
  Pd u = b-a, v = c-a, n = u/v;
  if (n.dist2() < EPS) return getC(a, b);
  Pd top = (n/u) * v.dist2() + (v/n) * u.dist2();
  return a + top / (2*n.dist2());
}
Pd getC(Pd a, Pd b, Pd c, Pd d) {
  Pd p = b-a, q = c-a, r = d-a;
  double bot = ((p / q) * r) * 2.0;
  if (abs(bot) < EPS) return a;
  Pd top = (p/q) * r.dist2() + (q/r) * p.dist2() +
  (r/p) * q.dist2();
  return a + top / bot;
} */
mt19937_64
rng(chrono::steady_clock::now().time_since_epoch().count()
pair<Pd,double> min_circle(vector<Pd> v) {
  shuffle(all(v), rng);
  Pd c = { 0, 0 }; double r = 0;
  auto dist = [&](Pd p1, Pd p2) { return
  sqrt((p1-p2).dist2()); };
  auto chk = [&](Pd p) { return dist(c, p) > r+EPS; };
  for (int i = 0; i < sz(v); i++) if (chk(v[i])) {
    c = v[i]; r = 0;
    for (int j = 0; j < i; j++) if (chk(v[j])) {
      c = getC(v[i], v[j]); r = dist(c, v[i]);
      for (int k = 0; k < j; k++) if (chk(v[k])) {
        c = getC(v[i], v[j], v[k]); r = dist(c, v[k]);
      }
    }
  }
  return { c, r };
}
```

6.10 Geometric Intersections

Usage: Intersection primitives: Segment, Line-Circle, Line-Hull, and Circle-Polygon area.
Time Complexity: $O(1)/O(1)/O(\log N)/O(N)$

```
using T = __int128_t; // T <= O(COORD^3)
// [1] 선분 교차 (정밀 좌표) / Param: 선분 ab, 선분 cd
// Return: {flag, xn, xd, yn, yd} -> 교점 (xn/xd,
yn/yd)
// Flag: 0(안만남), 1(교점=끝점), 4(교차), -1(무수히 겹
침)
tuple<int, T, T, T, T> segmentInter(P a, P b, P c, P d)
{
  if (!isInter(a, b, c, d)) return {0, 0, 0, 0, 0};
  T det = (b - a) / (d - c);
  if (det == 0) {
    if (b < a) swap(a, b);
    if (d < c) swap(c, d);
    if (b == c) return {1, b.x, 1, b.y, 1};
    if (d == a) return {1, d.x, 1, d.y, 1};
    return {-1, 0, 0, 0, 0};
  }
  T p = (c - a) / (d - c), q = det;
  T xp = a.x * q + (b.x - a.x) * p, xq = q;
  T yp = a.y * q + (b.y - a.y) * p, yq = q;
  if (xq < 0) { xp = -xp; xq = -xq; }
  if (yq < 0) { yp = -yp; yq = -yq; }
  T g_x = __gcd(xp, xq); xp /= g_x; xq /= g_x;
  T g_y = __gcd(yp, yq); yp /= g_y; yq /= g_y;
  int f = 4;
  if ((xp == a.x * xq && yp == a.y * yq) || (xp == b.x
  * xq && yp == b.y * yq)) f = 1;
  if ((xp == c.x * xq && yp == c.y * yq) || (xp == d.x
  * xq && yp == d.y * yq)) f = 1;
  return { f, xp, xq, yp, yq };
}
// [2] 원-직선 교차 / Param: 직선 ab, 원(c, r)
// Return: 교점 좌표 목록 (a -> b 방향 순서 정렬됨)
vector<Pd> lineCircle(P a, P b, P c, double r) {
  P ab = b - a;
  Pd p = a.d() + ab.d() * ((c - a) * ab /
  (double)ab.dist2());
  double h2 = r * r - (p - c.d()).dist2();
  if (h2 < -EPS) return {};
  if (abs(h2) < EPS) return {p};
  Pd h = ab.d() * (sqrt(h2) / sqrt(ab.dist2()));
  return {p - h, p + h};
}
// [3] 원-다각형 교차 넓이
// Param: 원(c, r), 다각형 poly (CCW/CW 무관) Return:
교차하는 영역의 넓이
double areaCirclePoly(P c, double r, const vector<P>&
poly) {
  auto tri = [&](Pd p, Pd q) {
    Pd d = q - p;
    double a = d * p / d.dist2(), b = (p.dist2() - r *
    r) / d.dist2();
    double det = a * a - b;
    if (det <= EPS) return r * r * atan2(p / q, p * q);
    double t1 = -a - sqrt(max(0.0, det)), t2 = -a +
    sqrt(max(0.0, det));
    if (t2 < -EPS || t1 > 1.0 + EPS) return r * r *
    atan2(p / q, p * q);
    Pd u = p + d * max(0.0, t1), v = p + d * min(1.0,
    t2);
    return r * r * atan2(p / u, p * u) + u / v + r * r
    * atan2(v / q, v * q);
  };
  double res = 0; int n = poly.size();
  for(int i = 0; i < n; i++)
    res += tri((poly[i] - c).d(), (poly[(i + 1) % n] -
    c).d());
  return res * 0.5;
}
// [4] 볼록 다각형-직선 교차 (O(log N)) Param: 볼록 다
각형 h (CCW), 직선 ab
// Return: {i, j} -> 직선이 변(i, i+1)과 변(j, j+1)을
교차함. 안만나면 {-1, -1}
int extrVertex(const vector<P>& h, P dir) {
  int n = h.size();
  auto cmp = [&](int i, int j) { return ccw({0,0}, dir,
  h[i%n] - h[j%n]); };
  auto isExtr = [&](int i) { return cmp(i, i - 1 + n)
  >= 0 && cmp(i, i + 1) > 0; };
  if (isExtr(0)) return 0;
  int l = 0, r = n;
  while (l + 1 < r) {
    int m = (l + r) / 2;
    if (isExtr(m)) return m;
    if (cmp(l + 1, l) > 0) {
      if (cmp(m + 1, m) > 0 && cmp(l, m) > 0) r = m;
      else l = m;
    } else {
      if (cmp(m + 1, m) <= 0 && cmp(l, m) < 0) l = m;
      else r = m;
    }
  }
  return l;
}
array<int,2> hullLineInter(const vector<P>& h, P a, P
b) {
  P dir = b - a, per = {-dir.y, dir.x};
  int n = h.size();
  int i1 = extrVertex(h, per), i2 = extrVertex(h, per *
  -1);
  if (ccw(a, b, h[i1]) * ccw(a, b, h[i2]) > 0) return
  {-1, -1};
  auto search = [&](int s, int e) {
    if (ccw(a, b, h[s]) == 0) return s;
    int l = s, r = e;
    if (r < l) r += n;
    while (l + 1 < r) {
      int m = (l + r) / 2;
      if (ccw(a, b, h[m % n]) == ccw(a, b, h[s])) l =
      m; else r = m;
    }
    return l % n;
  };
  return {search(i1, i2), search(i2, i1)};
}
```

6.11 Half Plane Intersection

Usage: Intersection of half-planes defined by lines (left side is valid). Returns a convex polygon.
Time Complexity: $O(N \log N)$

```
// 각 선분의 '왼쪽' 영역들의 교집합(볼록 다각형)을 반환
// 영역이 없거나 닫히지 않는 경우(Unbounded) 빈 벡터 반
환 가능성 있음
struct Line {
  double a, b, c; // ax + by <= c
  Line(Pd p1, Pd p2) {
    a = p1.y - p2.y; // -dy
    b = p2.x - p1.x; // dx
    c = a * p1.x + b * p1.y;
  }
  Pd slope() const { return {a, b}; }
};
Pd intersect(Line u, Line v) { // 평행하지 않은 두 직선
의 교점
  double det = u.a * v.b - u.b * v.a;
  return {(u.c * v.b - u.b * v.c) / det, (u.a * v.c -
  u.c * v.a) / det};
}
bool bad(Line l, Pd p) {
  return l.a * p.x + l.b * p.y > l.c + EPS;
}
vector<Pd> HPI(vector<Line> lines) {
  sort(all(lines), [&](const Line& u, const Line& v) {
    Pd p1 = u.slope(), p2 = v.slope();
    bool f1 = p1.y > 0 || (p1.y == 0 && p1.x > 0);
    bool f2 = p2.y > 0 || (p2.y == 0 && p2.x > 0);
```

```
    if (f1 != f2) return f1 > f2;
    if (abs(p1 / p2) > EPS) return (p1 / p2) > 0;
    return u.c < v.c;
  });
  deque<Line> dq;
  for (auto& l : lines) {
    if (!dq.empty() && abs(dq.back().slope() /
    l.slope()) < EPS) continue;
    while (sz(dq) >= 2 && bad(l, intersect(dq.back(),
    dq[sz(dq)-2]))) dq.pop_back();
    while (sz(dq) >= 2 && bad(l, intersect(dq[0],
    dq[1]))) dq.pop_front();
    dq.push_back(l);
  }
  while (sz(dq) > 2 && bad(dq[0], intersect(dq.back(),
  dq[sz(dq)-2]))) dq.pop_back();
  while (sz(dq) > 2 && bad(dq.back(), intersect(dq[0],
  dq[1]))) dq.pop_front();
  vector<Pd> res; if (sz(dq) < 3) return {};
  for (int i = 0; i < sz(dq); i++) {
    res.push_back(intersect(dq[i], dq[(i + 1) %
    sz(dq)]));
  }
  return res;
}
```

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 {
      int nxt[26], fail, link = -1, id = -1;
      Node() { memset(nxt, 0, sizeof nxt); }
  };
  vector<Node> trie;
  AhoCorasick() { trie.emplace_back(); }
  void insert(const string& s, int id) {
      int u = 0;
      for (char c : s) {
          int &v = trie[u].nxt[c - 'a'];
          if (!v) v = trie.size(), trie.emplace_back();
          u = v;
      }
      trie[u].id = id;
  }
  void build() {
      queue<int> q;
      for (int i = 0; i < 26; i++) if (trie[0].nxt[i])
      q.push(trie[0].nxt[i]);
      while (!q.empty()) {
          int u = q.front(); q.pop();
          for (int i = 0; i < 26; i++) {
              int &v = trie[u].nxt[i], f =
              trie[trie[u].fail].nxt[i];
              if (v) {
                  trie[v].fail = f;
                  trie[v].link = (trie[f].id != -1) ? f
                  : trie[f].link;
                  q.push(v);
              } else v = f;
          }
      }
  }
  vector<pair<int,int>> query(const string& s) {
      vector<pair<int,int>> res;
      int u = 0;
      for (int i = 0; i < (int)s.size(); i++) {
          u = trie[u].nxt[s[i] - 'a'];
          for (int t = u; t != -1; t = trie[t].link) {
              if (trie[t].id != -1) res.emplace_back(i,
              trie[t].id);
          }
      }
      return res;
  }
};
int main() {
  AhoCorasick ac;
  vector<string> patterns = {"he", "she", "hers",
  "his"};
  for(int i = 0; i < sz(patterns); 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: $O(N + M)$

```
template <typename T>
struct KMP {
  T P; vector<int> pi;
  KMP(const T& P) : P(P), pi(sz(P)) {
    for (int i = 1, j = 0; i < sz(P); i++) {
      while (j > 0 && P[i] != P[j]) j = pi[j-1];
      if (P[i] == P[j]) pi[i] = ++j;
    }
  }
  vector<int> find(const T& S) {
    vector<int> res;
    int n = sz(S), m = sz(P), j = 0;
    for (int i = 0; i < n; i++) {
      while (j > 0 && S[i] != P[j]) j = pi[j-1];
      if (S[i] == P[j]) {
        if (j == m-1) {
          res.push_back(i-m+1); j = pi[j];
        }
        else j++;
      }
    }
    return res;
  }
  int minPeriod() {
    int m = sz(P); if (m == 0) return 0;
    int len = m - pi[m-1]; if (m % len == 0) return
    len;
```

```
    return m;
  }
};
```

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.
Time Complexity: $O(N \log N)$

```
// calculates suffix array with O(n*logn)
auto get_sa(const string& s) {
  const int n = s.size(), m = max(256, n) + 1;
  vector<int> sa(n), r(n << 1), nr(n << 1), cnt(m),
  idx(n);
  for (int i = 0; i < n; i++) sa[i] = i, r[i] = s[i];
  for (int d = 1; d < n; d <<= 1) {
    auto cmp = [&](int a, int b) { return r[a] < r[b]
    || r[a] == r[b] && r[a + d] < r[b + d];};
    for (int i = 0; i < m; ++i) cnt[i] = 0;
    for (int i = 0; i < n; ++i) cnt[r[i + d]]++;
    for (int i = 1; i < m; ++i) cnt[i] += cnt[i - 1];
    for (int i = n - 1; ~i; --i) idx[--cnt[r[i + d]]] =
    i;
    for (int i = 0; i < m; ++i) cnt[i] = 0;
    for (int i = 0; i < n; ++i) cnt[r[i]]++;
    for (int i = 1; i < m; ++i) cnt[i] += cnt[i - 1];
    for (int i = n - 1; ~i; --i) sa[--cnt[r[idx[i]]]] =
    idx[i];
    nr[sa[0]] = 1;
    for (int i = 1; i < n; ++i) nr[sa[i]] = nr[sa[i -
    1]] + cmp(sa[i - 1], sa[i]);
    for (int i = 0; i < n; ++i) r[i] = nr[i];
    if (r[sa[n - 1]] == n) break;
  }
  return sa;
}
// calculates lcp array. it needs suffix array &
original sequence with O(n)
auto get_lcp(const string& s, const auto& sa) {
  const int n = s.size(); vector lcp(n - 1, 0), isa(n,
  0);
  for (int i = 0; i < n; i++) isa[sa[i]] = i;
  for (int i = 0, k = 0; i < n; i++) if (isa[i]) {
    for (int j = sa[isa[i] - 1]; s[i + k] == s[j + k];
    k++);
    lcp[isa[i] - 1] = k ? k-- : 0;
  }
  return 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;
}
```

7.7 Eertree

Usage: Manages all distinct palindromic substrings using length and suffix links.
Time Complexity: $O(N \log \Sigma)$

```
// 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;
}
```

7.8 Suffix Automaton

Usage: Builds a DFA representing all substrings. Supports substring counting and pattern matching.
Time Complexity: $O(N \cdot \Sigma)$

```
template<typename T, size_t S, T init_val>
struct initialized_array : public array<T, S> {
  initialized_array(){ this->fill(init_val); }
};
template<class Char_Type, class Adjacency_Type>
struct suffix_automaton{
  // Begin States
  // len: length of the longest substring in the class
  // link: suffix link
  // firstpos: minimum value in the set endpos
  vector<int> len{0}, link{-1}, firstpos{-1},
  is_clone{false};
  vector<Adjacency_Type> next{{}};
  ll ans{0LL}; // 서로 다른 부분 문자열 개수
  // End States
  void set_link(int v, int lnk){
    if(link[v] != -1) ans -= len[v] - len[link[v]];
    link[v] = lnk;
    if(link[v] != -1) ans += len[v] - len[link[v]];
  }
  int new_state(int l, int sl, int fp, bool c, const
  Adjacency_Type &adj){
    int now = len.size(); len.push_back(l);
    link.push_back(-1);
    set_link(now, sl); firstpos.push_back(fp);
    is_clone.push_back(c); next.push_back(adj); return
    now;
  } int last = 0;
  void extend(const vector<Char_Type> &s){
    last = 0; for(auto c: s) extend(c); }
  void extend(Char_Type c){
    int cur = new_state(len[last] + 1, -1, len[last],
    false, {}), p = last;
    while(~p && !next[p][c]) next[p][c] = cur, p =
    link[p];
    if(!~p) set_link(cur, 0);
    else{
```

```
      int q = next[p][c];
      if(len[p] + 1 == len[q]) set_link(cur, q);
      else{
        int clone = new_state(len[p] + 1, link[q],
        firstpos[q], true, next[q]);
        while(~p && next[p][c] == q) next[p][c] =
        clone, p = link[p];
        set_link(cur, clone); set_link(q, clone);
      }
    }
    last = cur;
  }
  int size() const {  return (int)len.size(); } // # of
  states
}; suffix_automaton<int, initialized_array<int,26,0>>
T;
// for(auto c : s) if((x=T.next[x][c]) == 0) return
false;
```

8 STL & pbds

8.1 Hash map (pb_ds)

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 (pb_ds)

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

```
// k번째 원소 확인 및 x보다 작은 원소개수 확인을
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.find_by_order(k): k번째 원소의 iterator 반환
(0-indexed, 없으면 os.end())
// os.order_of_key(x) : x보다 작은 원소의 개수 반환
template <typename T>
using ordered_multiset = tree<T, null_type,
less_equal<T>, rb_tree_tag,
tree_order_statistics_node_update>;
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();
} // os.erase(m_find(os, val))
```

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 (pb_ds)

Usage: Meldable heap supporting modify/erase via point_iterator.
Time Complexity: $O(\log N)$

```
// 큐 병합, 임의 값 수정 및 삭제 가능
#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: Persistent sequence supporting fast insertion, deletion and slicing.
Time Complexity: $O(\log N)$

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

8.6 Trie (pb_ds)

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.

```
#include <unistd.h>
constexpr int rbuf_sz = 1 << 20, 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.

```
#include <random>
#include <chrono>
mt19937_64
rng(chrono::steady_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%
```

9.4 Ternary Search

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

```
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; // for max: f(p) > f(q)
    else lo = p;
  }
  ll res = lo;
  for (ll i = lo+1; i <= hi; i++) if (f(i) < f(res))
  res = i;
  return idx;
}
double ternary_search(double lo, double hi, auto f, int
it=100) {
  while (it--) {
    double p = (lo*2 + hi) / 3., q = (lo + hi*2) / 3.;
    if (f(p) < f(q)) hi = q; // for max: f(p) > f(q)
    else lo = p;
  }
  return (lo+hi) / 2.;
}
```

9.5 Some tricks

Usage: Collection of bitwise hacks and optimization techniques.

```
__builtin_popcount(x); // 켜진 비트(1)의 총 개수
__builtin_clz(x); // 왼쪽(MSB)부터 연속된 0의 개수
__builtin_ctz(x); // 오른쪽(LSB)부터 연속된 0의 개수
// popcount를 유지하면서 다음으로 큰 수
bool next_combination(ll& bit, int N) {
  ll x = bit & -bit, y = bit + x;
  bit = (((bit & ~y) / x) >> 1) | y;
  return (bit < (1LL << N));
}
// v(>0)보다 크고 popcount가 같은 가장 작은 정수
ll next_perm(ll v) {
  ll t = v | (v - 1);
  return (t + 1) | (((~t & -~t) - 1) >>
  (__builtin_ctz(v) + 1));
}
// mask의 모든 부분집합을 내림차순으로 순회 (0 제외),
O(3^N)
for (int submask = mask; submask > 0; submask =
(submask-1) & mask);
// mask를 포함하는 모든 상위집합을 오름차순으로 순회
for (int supmask = mask; supmask < (1 << n); supmask =
(supmask+1) | mask);
// 런타임 변수 n에 맞는 크기의 bitset을 사용
const int MAXLEN = 200005; // 최대 범위
template <int len = 1>
void solve(int n) {
  if (len < n) { solve<min(len * 2, MAXLEN)>(n);
  return; }
  bitset<len> bs;
  // do stuff
}
// bitset 고속순회 (켜져있는 비트만 순회)
void bitset_iterate(bitset<1000>& bs) {
  int idx = bs._Find_first();
  while (idx < bs.size()) {
    // do stuff
    idx = bs._Find_next(idx);
  }
}
// 1부터 n까지의 수에서 숫자 i가 등장하는 총 횟수
ll count_digit_frq(ll n, int i) {
  ll ret = 0;
  for (ll j = 1; j <= n; j *= 10) {
    ll div = j * 10, quote = n / div, rem = n % div;
    if (i == 0) ret += (quote - 1) * j;
    else ret += quote * j;
    if (rem >= i * j) {
      if (rem < (i + 1) * j) ret += rem - i * j + 1;
      else ret += j;
```

```
    }
  }
  return ret;
}
// 특정 날짜(년, 월, 일)의 요일 / 0: Sat, 1: Sun, ...
int get_day_of_week(int y, int m, int d) {
  if (m <= 2) y--, m += 12; int c = y / 100; y %= 100;
  int w = ((c>>2)-(c<<1)+y+(y>>2)+(13*(m+1)/5)+d-1) %
  7;
  if (w < 0) w += 7; return w;
}
```

10 Checklist + Useful Info

10.1 Highly Composite Numbers, Large Prime

```
< 10^k             number    divs  2 3 5 71113171923293137
-----------------------------------------------------------
1                       6       4  1 1
2                      60      12  2 1 1
3                     840      32  3 1 1 1
4                    7560      64  3 3 1 1
5                   83160     128  3 3 1 1 1
6                  720720     240  4 2 1 1 1 1
7                 8648640     448  6 3 1 1 1 1
8                73513440     768  5 3 1 1 1 1 1
9               735134400    1344  6 3 2 1 1 1 1
10             6983776800    2304  5 3 2 1 1 1 1 1
11            97772875200    4032  6 3 2 2 1 1 1 1
12           963761198400    6720  6 4 2 1 1 1 1 1 1
13          9316358251200   10752  6 3 2 1 1 1 1 1 1 1
14         97821761637600   17280  5 4 2 2 1 1 1 1 1 1
15        866421317361600   26880  6 4 2 1 1 1 1 1 1 1 1
16       8086598962041600   41472  8 3 2 2 1 1 1 1 1 1 1
17      74801040398884800   64512  6 3 2 2 1 1 1 1 1 1 1 1
18     897612484786617600  103680  8 4 2 2 1 1 1 1 1 1 1 1
```

```
< 10^k   prime  # of prime     < 10^k              prime
-----------------------------------------------------------
1            7          4      10             9999999967
2           97         25      11            99999999977
3          997        168      12           999999999989
4         9973       1229      13          9999999999971
5        99991       9592      14         99999999999973
6       999983      78498      15        999999999999989
7      9999991     664579      16       9999999999999937
8     99999989    5761455      17      99999999999999997
9    999999937   50847534      18     999999999999999989
```

10.2 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. - Misere Nim : 모든 돌 무더기가 1이면 N이 홀수일 때 후공 승, 그렇지 않은 경우 XOR 합 0이면 후공 승
- Pick' s Theorem
  격자점으로 구성된 simple polygon이 주어짐. I 는 polygon 내부의 격자점 수, B 는 polygon 선분 위 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) ≤ (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.

10.3 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)
- 구간을 통째로 가져간다 : 플로우 + 적당한 자료구조 $(i, i+1, k, 0), (s, e, 1, w), (N, T, k, 0)$
- 말도 안 되는 것 / 당연하다고 생각한 것 다시 생각해 보기
- 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

10.4 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
- $a \le b \le c \le d$에서 $A[a, c] + A[b, d] \le A[a, d] + A[b, c]$
- 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

10.5 Graph Matching(Graph with $|V| \leq 500$)

- **Game on a Graph** : s에 토큰이 있음. 플레이어는 각자의 턴마다 토큰을 인접한 정점으로 옮기고 못 옮기면 짐.
  s를 포함하지 않는 최대 매칭이 존재함 $\leftrightarrow$ 후공이 이김
- **Chinese Postman Problem** : 모든 간선을 방문하는 최소 가중치 Walk를 구하는 문제.
  Floyd를 돌린 다음, 홀수 정점들을 모아서 최소 가중치 매칭 (홀수 정점은 짝수 개 존재)
- **Unweighted Edge Cover** : 모든 정점을 덮는 가장 작은 (minimum cardinality/weight) 간선 집합을 구하는 문제
  $|V| - |M|$, 길이 3짜리 경로 없음, star graph 여러 개로 구성
- **Weighted Edge Cover** : $sum_{v \in V}(w(v)) - sum_{(u,v) \in M}(w(u) + w(v) - d(u,v))$, $w(x)$는 x와 인접한 간선의 최소 가중치
- **NEERC'18 B** : 각 기계마다 2명의 노동자가 다뤄야 하는 문제.
  기계마다 두 개의 정점을 만들고 간선으로 연결하면 정답은 $|M| - |\text{기계}|$임. 정답에 1/2씩 기여한다는 점을 생각해보면 좋음.
- **Min Disjoint Cycle Cover** : 정점이 중복되지 않으면서 모든 정점을 덮는 길이 3 이상의 사이클 집합을 찾는 문제.
  모든 정점은 2개의 서로 다른 간선, 일부 간선은 양쪽 끝점과 매칭되어야 하므로 플로우를 생각할 수 있지만 용량 2짜리 간선에 유량을 1만큼 흘릴 수 있으므로 플로우는 불가능.
  각 정점과 간선을 2개씩 $((v, v'), (e_{i,u}, e_{i,v}))$로 복사하자. 모든 간선 $e = (u, v)$에 대해 e_u와 e_v를 잇는 가중치 w짜리 간선을 만들고(like NEERC18), $(u, e_{i,u}), (u', e_{i,u}), (v, e_{i,v}), (v', e_{i,v})$를 연결하는 가중치 0짜리 간선을 만들자. Perfect 매칭이 존재함 $\Leftrightarrow$ Disjoint Cycle Cover 존재. 최대 가중치 매칭 찾은 뒤 모든 간선 가중치 합에서 매칭 빼면 됨.
- **Two Matching** : 각 정점이 최대 2개의 간선과 인접할 수 있는 최대 가중치 매칭 문제.
  각 컴포넌트는 정점 하나/경로/사이클이 되어야 함. 모든 서로 다른 정점 쌍에 대해 가중치 0짜리 간선 만들고, 가중치 0짜리 (v, v') 간선 만들면 Disjoing Cycle Cover 문제가 됨. 정점 하나만 있는 컴포넌트는 self-loop, 경로 형태의 컴포넌트는 양쪽 끝점을 연결한다고 생각하면 편함.

10.6 Mincut 모델링

- N개의 boolean 변수 $v_1, \cdots, v_n$을 정해서 비용을 최소화하는 문제=true인 점은 T, false인 점은 F와 연결되게 분할하는 민컷 문제
  1. v_i가 T일 때 비용 발생: i에서 F로 가는 비용 간선
  2. v_i가 F일 때 비용 발생: i에서 T로 가는 비용 간선
  3. v_i가 T이고 v_j가 F일 때 비용 발생: i에서 j로 가는 비용 간선
  4. $v_i \neq v_j$일 때 비용 발생: i에서 j로, j에서 i로 가는 비용 간선
  5. v_i가 T면 v_j도 T여야 함: i에서 j로 가는 무한 간선
  6. v_i가 F면 v_j도 F여야 함: j에서 i로 가는 무한 간선
- 5/6번 + v_i와 v_j가 달라야 한다는 조건이 있으면 MAX-2SAT
- Maximum Density Subgraph (NEERC'06H, BOJ 3611 팀의 난이도)
  1. density $\geq x$인 subgraph가 있는지 이분 탐색
  2. 정점 N개, 간선 M개, 차수 D_i개
  3. 그래프의 간선마다 용량 1인 양방향 간선 추가
  4. 소스에서 정점으로 용량 M, 정점에서 싱크로 용량 $M - D_i + 2x$
  5. min cut에서 S와 붙어 있는 애들이 1개 이상이면 x 이상이고, 그게 subgraph의 정점들
  6. while(r-l $\geq$ 1.0/(n*n)) 으로 해야 함. 너무 많이 돌리면 실수 오차