Índice

1. Setup

Template

```
#include <bits/stdc++.h>
using namespace std;
#define endl          '\n'
#define forn(i,N)     for (int i = 0; i < int(N); i++)
#define all(v)        begin(v), end(v)
#define dbg(x)        cerr << #x << " = " << (x) << endl
#define raya          cerr << "================" << endl
#define forall(it,v) for (auto it = begin(v); it != end(v); it++)
#define printall(v)  forall(x,v) { cout << *x << " "; } cout << endl
#define printpair(p) cout << "(" << p.first << ", " << p.second << ")
    ↪ " << endl

int main () { ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);



    return 0;
}
```

Makefile

```
CC = g++
CPPFLAGS = -Wall -g -fsanitize=undefined -fsanitize=bounds \
-std=c++17 -O0
```

comp.sh: Compilar $1 y mostrar primeras $2 lineas de error

```
clear
make -s $1 2>&1 | head -$2
```

run.sh: Correr $1 con el input $2

```
clear
make -s $1 && ./$1 < $2
```

2. STL

2.1. <algorithm.h>

Funciones que modifican rangos

Función	Params	Ejemplo
copy	first last result	`B.resize(A.size()); copy(all(A), B)`
fill	first last val	`memo.resize(MAXN); fill(all(memo), -1)`
rotate	first middle last	`rotate(begin(A), begin(A) + 3, end(A));`

Búsqueda binaria en vector ordenado

```cpp
template <class T> int primer_igual (vector<T>& arr, T x) {
    auto it = lower_bound(all(arr), x);
    if (it == arr.end() || *it != x) return -1;
    return it - arr.begin();
}

template <class T> int ultimo_igual (vector<T>& arr, T x) {
    if (arr.begin() == arr.end()) return -1;
    auto it = prev(upper_bound(all(arr), x));
    if (*it != x) return -1;
    return it - arr.begin();
}

template <class T> int ultimo_menor (vector<T>& arr, T x) {
    if (arr.begin() == arr.end()) return -1;
    auto it = prev(lower_bound(all(arr), x));
    if (*it >=) return -1;
    return it - arr.begin();
}

template <class T> int primer_mayor (vector<T>& arr, T x) {
    auto it = upper_bound(all(arr), x);
    if (it == arr.end()) return -1;
    return it - arr.begin();
}
```

Operaciones de conjuntos con vectors ordenados (lineal)

```cpp
// Siempre hacer resize al final asi:

vector<int> A = { 5, 10, 15, 20, 25};
vector<int> B = {10, 20, 30, 40, 50};

vector<int> U(A.size() + B.size());

auto it = set_union(all(A), all(B), begin(U));

U.resize(it - U.begin());
```

Función	Descripción
set_union	Unión
set_intersection	Intersección
set_difference	Elementos que están en el primero y no en el segundo
set_symmetric_difference	Elementos que están en uno pero no los dos (como el xor)

2.2. Set

Indexed set y multiset

```cpp
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

template<class T> struct IndexedSet {
    tree<
        T, null_type, less<T>,
        rb_tree_tag, tree_order_statistics_node_update
    > s;
    void add  (T   x) { ms.insert(x); }
    int  idx  (T   x) { return  ms.order_of_key(x); }
    bool has  (T   x) { return  ms.find(x) != ms.end(); }
    T    ith  (int i) { return *ms.find_by_order(i); }
};

template<class T> struct IndexedMultiset {
    int t = 0; tree<
        pair<T, int>, null_type, less<pair<T, int>>,
        rb_tree_tag, tree_order_statistics_node_update
    > ms;
    void add  (T   x) { ms.insert(mp(x, t++)); }
    int  nle  (T   x) { return ms.order_of_key(mp(x, -1)); }
    int  nleq (T   x) { return ms.order_of_key(mp(x, INT_MAX)); }
    int  cnt  (T   x) { return nleq(x) - nle(x); }
    T    ith  (int i) { return (*ms.find_by_order(i)).fst; }
};
```

Intervalos consecutivos

```cpp
struct IntervalosConsecutivos {
    set<int> I;
    map<int, int> L;
    IntervalosConsecutivos (int i, int j) {
        I.insert(i);
        I.insert(j);
        L[j - i]++;
    }
    void cortar (int k) {
        int i = *prev(I.lower_bound(k));
        int j = *(I.lower_bound(k));
        L[j - i]--;
        if (L[j - i] == 0) L.erase(j - i);
        L[k - i]++;
        L[j - k]++;
        I.insert(k);
    }
    int max_intervalo () {
        return (*L.rbegin()).fst;
    }
};
```

2.3. TODO

- Priority queue custom compare

3. Range queries

Prefix/dff arrays

```
// Usar array indexado desde 1 con A[0] = 0
// Usar intervalos cerrado-cerrado (indexados desde 1)

using ll = long long;

vector<ll> derivada (vector<ll>& A) {
    vector<ll> D(A.size());
    forn(i, A.size() - 1) D[i] = A[i+1] - A[i];
    return D;
}

void derivada_range_update (vector<ll>& D, int i, int j, ll v) {
    D[i-1] += v;
    D[j]   -= v;
}

vector<ll> integral (vector<ll>& A) {
    vector<ll> I(A.size() + 1);
    I[0] = 0;
    forn(i, A.size()) I[i+1] = I[i] + A[i];
    return I;
}

ll integral_range_query (vector<ll>& I, int i, int j) {
    return I[j+1] - I[i];
}
```

Segment tree range query point set

```
template<class T> struct SegTree {
    vector<T>& arr; int N;
    // Elegir operacion y neutro
    T id;
    T op (T a, T b) { return 0; }
    vector<T> t;
    void make () {
        t.resize(N << 1); forn(i,N) t[i+N] = arr[i];
        for (int i = N - 1; i; i--) t[i] = op(t[i<<1], t[i<<1|1]);
    }
    void set (int i, T v) {
        for(t[i += N] = v; i > 1; i >>= 1) t[i>>1] = op(t[i], t[i^1])
            ↪ ;
    }
    T q (int l, int r) {
        T res = id;
        for (l += N, r += N; l < r; l >>= 1, r >>= 1) {
            if (l&1) res = op(res, t[l++]);
            if (r&1) res = op(res, t[--r]);
        } return res;
    }
};

// Usar asi:
vector<int> A = {...};

SegTree<int> segtree = {A, A.size(), 0};
segtree.make();
```

Sparse table

```
// Operacion asociativa IDEMPOTENTE

#define log2fl(x) (x ? 63 - __builtin_clzll(x) : -1)

struct STable {
    vector<int>& arr; int N;
    vector<vector<int>> st;
    // Modificar operacion
    int op (int a, int b) { return min(a,b); }
    void make () {
        st.resize(20, vector<int>(N));
        st[0] = arr; scn(w,1,19) scn(i,0,N - (1 << w))
            st[w][i] = op(st[w-1][i], st[w-1][i + (1 << (w-1))]);
    }
    int q (int i, int j) {
        int w = log2fl(j - i + 1);
        return op(st[w][i], st[w][j - (1 << w) + 1]);
    }
};

// Usar asi:
vector<int> A = {...};

STable stable = {A, A.size()};
stable.make();
```

3.1. TODO

- Prefix/diff arrays 2D
- Fenwick 1D (base + RQPURUPQRQRU)
- Fenwick 2D (base + RQPURUPQRQRU)

4. Matemática

4.1. Aritmética

Techo de la división

```
#define ceildiv(a,b) ((a + b - 1) / b)
```

Piso de la raiz cuadrada

```
using ll = long long;

ll isqrt (ll x) {
    ll s = 0;
    for (ll k = 1 << 30; k; k >>= 1)
        if ((s+k) * (s+k) <= x) s += k;
    return s;
}
```

Piso del log2

```
#define log2fl(x) (x ? 63 - __builtin_clzll(x) : -1)
```

Aritmética en Zp

```
using ll = long long;

const ll mod = 1e9 + 7;

ll resta_mod (ll a, ll b) { return (a - b + mod) % mod; }

ll pow_mod (ll x, ll n) {
    ll res = 0;
    while (n) {
        if (n % 2) res = res * x % mod;
        n /= 2;
        x = x * x % mod;
    } return res;
}

ll div_mod (ll a, ll b) { return a * pow_mod(b, mod - 2) % mod; }
```

4.2. Sin categorizar

Test de primalidad

```
struct Primetest {
    bool c[1000001]; vector<int> p;
    primetest () {
        p.reserve(1<<16); scn(i,2,1000000) if (!c[i]) {
            p.pb(i); for (int j = 2; i*j < 1000001; j++) c[i*j] = 1;
        }
    }
    bool isprime (int x) {
        for (int i = 0, d = p[i]; d*d <= x; d = p[++i])
            if (!(x % d)) return false;
        return x >= 2;
    }
};
```

Template geometría

```
using flt = long double;
const flt EPS = 1e-9;
bool  flt_leq (flt a, flt b) { return a < b + EPS; }
bool  flt_eq  (flt a, flt b) { return -EPS <= a - b && a - b <= EPS;
    ↪ }

struct Vec {
    int x, y;
    Vec operator+(Vec p) { return {x + p.x, y + p.y}; }
    Vec operator-(Vec p) { return {x - p.x, y - p.y}; }
    int operator*(Vec p) { return x * p.x + y * p.y; }
    int operator^(Vec p) { return x * p.y + y * p.x; }
};

int norma2 (Vec p) { return p.x * p.x + p.y * p.y; }
```

5. Sin categorizar

Búsqueda binaria sobre un predicado

```
using ll = long long;

// Si existe, el primer i donde pred(i) == true
// Si es todo false, devuelve d

ll bsearch (ll i, ll j, bool (*pred)(ll), ll d) {
    while (!(i + 1 == j)) {
        ll m = i + ((j - i) >> 1);
        pred(m) ? j = m : i = m;
    }
    if (pred(i)) return i;
    if (pred(j)) return j;
    return d;
}
```

Enumerar subconjuntos de un conjuto con bitmask

```
// Imprimir representaciones en binario de todos los numeros "[0,
    ↪ ..., 2^N-1]"
```

```
forn(mask, (1 << N)) {
    forn(i, N) cout << "01"[(mask & (1 << i)) > 0] << "\0\n"[i == N
        ↪ -1];
}

// Iterar por los bits de cada subconjunto
forn(mask, (1 << N)) {
    forn(i, N) {
        bool on = (mask & (1 << i)) > 0;
        if (on) { ... }
        else { ... }
    }
}
```

Disjoint set union

```
struct DSU {
    vector<int> p, w; int nc;
    DSU (int n) {
        nc = n, p.resize(n), w.resize(n);
        forn(i,n) p[i] = i, w[i] = 1;
    }
    int get (int x) { return p[x] == x ? x : p[x] = get(p[x]); }
    void join (int x, int y) {
        x = get(x), y = get(y);
        if (x == y) return;
        if (w[x] > w[y]) swap(x,y);
        p[x] = y, w[y] += w[x];
    }
    bool existe_camino (int x, int y) { return get(x) == get(y); }
};

// Usar asi:
int N = ...;
DSU dsu(N);
```

Hashing Rabin Karp

```
using ll = long long;

const ll primo = 27, MAX_PRIME_POW = 1e6;

ll prime_pow[MAX_PRIME_POW];
void get_prime_pow () {
    prime_pow[0] = 1;
    forn(i, MAX_PRIME_POW) prime_pow[i+1] = prime_pow[i] * primo %
        ↪ mod;
}

vector<ll> get_rolling_hash (string& s) {
    vector<ll> rh(s.size() + 1);
    rh[0] = 0;
    // Ojo: es 'A' o 'a' ???
    forn(i, s.size()) rh[i+1] = (rh[i] * primo % mod + s[i] - 'A') %
        ↪ mod;
    return rh;
}

ll hash_range_query (vector<ll>& rh, int i, int j) {
    j++;
    return (rh[j] - (rh[i] * prime_pow[j - i] % mod) + mod) % mod;
}
```

6. Brainstorming

- Graficar como puntos/grafos
- Pensarlo al revez
- ¿Que propiedades debe cumplir una solución?
- Si existe una solución, ¿existe otra más simple?
- ¿Hay elecciones independientes?
- ¿El proceso es parecido a un algoritmo conocido?
- Si es busca calcular $f(x)$ para todo x, calcular cuánto contribuye x a $f(y)$ para los otros y