# Índice



# 1. Preámbulo

## 1.1. Template

```cpp
#include <bits/stdc++.h>
using namespace std;
using i64 = int64_t;
#define rep(i,N)   for (int i = 0; i < int(N); i++)
#define scn(k,i,j) for (int k = int(i); k <= int(j); k++)
#define pb         push_back
#define endl       '\n'
// Pair
#define mp  make_pair
#define fst first
#define snd second
// Print
#define forall(it,v) for(auto it = v.begin(); it != v.end(); it++)
#define printall(v)  forall(x,v){cout << *x << " ";} cout << endl
#define printpair(p) cout << "(" << p.fst << ", " << p.snd << ")" <<
    endl

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

    // :)

    return 0;
}
```

## 1.2. Shell

```
// Makefile
CPPFLAGS = -std=c++20 -O0 -Wall -g
CC = g++
// comp.sh: compilar $1 y mostrar primeras $2 lineas de error
rm -f $1
clear
make $1 2>&1 | head -$2
// run.sh: correr $1 con $2 como input
rm -f $1
clear
make $1 && ./$1 < $2
```

# 2. STL

## 2.1. Resumen

| Función | Params | Descripción |
|---|---|---|
| assign | first last / n val | resize y asignación |
| find | first last val | primer = |
| is_sorted | first last comp | true si esta ordenado |
| sort, stable_sort | first last comp | ordena el intervalo |
| binary_search | first last val comp | true si aparece |
| lower_bound | first last val comp | primer >= |
| upper_bound | first last val comp | primer > |
| next_permutation | first last | sort; do {...} while (next_perm); |
| prev_permutation | first last | sort; reverse; do {...} while (...); |
| lexicographical_compare | first last 1,2 comp | "aabbcc" < "aabc" |

## 2.2. Order statistics multiset

**Warning: es pesado.** Ver time limit del problema.

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace  __gnu_pbds;
struct osms {
    int t = 0; tree<
        pair<int,int>, null_type, less<pair<int,int>>,
        rb_tree_tag, tree_order_statistics_node_update
    > ms;
    void add  (int x) { ms.insert(mp(x,t++)); }
    int  nle  (int x) { return ms.order_of_key(mp(x,-1)); }
    int  nleq (int x) { return ms.order_of_key(mp(x,INT_MAX)); }
    int  cnt  (int x) { return nleq(x) - nle(x); }
    int  ith  (int i) { return (*ms.find_by_order(i)).fst; }
    int size  (void)  { return ms.size(); }
};
```

# 3. Estructuras

## 3.1. Prefix table

`PTable pt = {arr, arr.size()}; pt.make();`

```
struct PTable {
    vector<i64> &arr; int N;
    vector<i64> pt;
    void make () {
        pt.resize(N);
        rep(i,N) pt[i] = !i ? arr[i] : pt[i-1] + arr[i];
    }
    i64 q (int i, int j) { return (--i < 0) ? pt[j] : pt[j] - pt[i]; }
};
```

## 3.2. Sparse table

Operacion asociativa **idempotente**.
`STable st = {arr, arr.size()}; st.make();`

```
struct STable {
    vector<int>& arr; int N;
    vector<vector<int>> st;
    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]);
    }
};
```

## 3.3. Segment tree (RMQ)

`SegTree<int>st = {arr, N}; st.make();`

```
template<class T> struct SegTree {
    vector<T>& arr; int N;
    T op (T a, T b) { return min(a,b); }
    T id = INT_MAX;
    vector<T> t;
    void make () {
        t.resize(N << 1); rep(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, int 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) {
        int 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;
    }
};
```

### 3.4. Disjiont set union

`DSU dsu(N);`

```
struct DSU {
    vector<int> p, w; int nc;
    DSU (int n) { nc = n; p.resize(n); w.resize(n); rep(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 same (int x, int y) { return get(x) == get(y); }
};
```

## 4. Algoritmos

### 4.1. Búsqueda binaria

Si existe, idx de primer `true` Si no, `i - 1`

```
i64 bsearch (i64 i, i64 j, bool (*pred)(i64)) {
    int d = i-1; while (!(i + 1 == j)) {
        i64 m = i + ((j - i) >> 1);
        pred(m) ? j = m : i = m;
    }
    if (pred(i)) return i;
    if (pred(j)) return j;
    return d;
}
```

## 5. Matemática

### 5.1. Cuentas

```
#define ceildiv(a,b) ((a+b-1)/b)
#define log2fl(x) (x ? 63 - __builtin_clzll(x) : -1)
```

### 5.2. Sqrt

```
i64 isqrt (i64 x) {
    i64 s = 0; for (i64 k = 1 << 30; k; k >>= 1)
        if ((s+k)*(s+k) <= x) s += k;
    return s;
}
```

### 5.3. Prime test

```
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;
    }
};
```

## 6. Grafos

### 6.1. Preámbulo

```
typedef vector<vector<int>> adj;
typedef vector<vector<pair<int,i64>>> wadj;
```

### 6.2. Euler tour

`ETour et = {G, N}; et.make(0);`

```cpp
struct ETour {
    vector<vector<int>>& adj; int N;
    vector<int> tour, first, depth;
    void dfs (int u, int d = 0) {
        depth[u] = d;
        first[u] = tour.size();
        tour.push_back(u);
        for (int v : adj[u]) { dfs(v,d+1); tour.push_back(u); }
    }
    void make (int r) {
        first.resize(N);
        depth.resize(N);
        dfs(r);
    }
};
```

### 6.3. LCA: O(N) preprocess

`LCA lca = {G, G.size()}; lca.make(root);`

```cpp
struct LCA {
    vec<vec<int>>& adj; int N, M;
    vec<int> tour, first, depth, tree;
    void dfs (int u, int d = 0) {
        depth[u] = d; first[u] = tour.size(); tour.pb(u);
        for (int v : adj[u]) { dfs(v,d+1); tour.pb(u); }
    }
    int op (int a, int b) {
        if (a == -1) return b;
        if (b == -1) return a;
        return depth[a] < depth[b] ? a : b;
    }
    int build (int v, int i, int j) {
        if (i == j) return tree[v] = tour[i];
        int m = (i + j) / 2;
        int lc = build(2*v, i, m);
        int rc = build(2*v+1, m+1, j);
        return tree[v] = op(lc,rc);
    }
    int query (int v, int i, int j, int ti, int tj) {
        if ( j <  ti || tj <   i) return -1;
        if (ti <=  i &&  j <= tj) return tree[v];
        int m = (i + j) / 2;
        int lc = query(2*v, i, m, ti, tj);
        int rc = query(2*v+1, m+1, j, ti, tj);
        return op(lc,rc);
    }
    void make (int r) {
        first.resize(N); depth.resize(N);
        dfs(r); M = tour.size();
        tree.assign(4*M, -1); build(1, 0, M-1);
    }
    int lca (int u, int v) {
        int i = first[u], j = first[v];
        if (i > j) swap(i,j);
        return query(1, 0, M-1, i, j);
    }
};
```

# 7. Misc.

## 7.1. Operaciones de bits

```cpp
#define bits(x) __builtin_popcount(x)
```