Índice

1. algorithm

#include <algorithm> #include <numeric>

Algo	**Params**	**Funcion**
sort, stable_sort	f, l	ordena el intervalo
nth_element	f, nth, l	*void* ordena el n-esimo, y particiona el resto
fill, fill_n	f, l / n, elem	*void* llena [f, l) o [f, f+n) con elem
lower_bound, upper_bound	f, l, elem	*it* al primer / ultimo donde se puede insertar elem para que quede ordenada
binary_search	f, l, elem	*bool* esta elem en [f, l)
copy	f, l, resul	hace resul+i=f+i $\forall i$
find, find_if, find_first_of	f, l, elem / pred / f2, l2	*it* encuentra i ∈[f,l) tq. i=elem, pred(i), i∈[f2,l2)
count, count_if	f, l, elem/pred	cuenta elem, pred(i)
search	f, l, f2, l2	busca [f2,l2) ∈ [f,l)
replace, replace_if	f, l, old / pred, new	cambia old / pred(i) por new
reverse	f, l	da vuelta
partition, stable_partition	f, l, pred	pred(i) ad, !pred(i) atras
min_element, max_element	f, l, [comp]	*it* min, max de [f,l]
lexicographical_compare	f1,l1,f2,l2	*bool* con [f1,l1]¡[f2,l2]
next/prev_permutation	f,l	deja en [f,l) la perm sig, ant
set_intersection, set_difference, set_union, set_symmetric_difference,	f1, l1, f2, l2, res	[res, . . .) la op. de conj
push_heap, pop_heap, make_heap	f, l, e / e /	mete/saca e en heap [f,l), hace un heap de [f,l)
is_heap	f,l	*bool* es [f,l) un heap
accumulate	f,l,i,[op]	$T = \sum$/oper de [f,l)
inner_product	f1, l1, f2, i	T = i + [f1, l1) . [f2, . . .)
partial_sum	f, l, r, [op]	r+i = $\sum$/oper de [f,f+i] $\forall i$ ∈[f,l)
__builtin_ffs	unsigned int	Pos. del primer 1 desde la derecha
__builtin_clz	unsigned int	Cant. de ceros desde la izquierda.
__builtin_ctz	unsigned int	Cant. de ceros desde la derecha.
__builtin_popcount	unsigned int	Cant. de 1's en x.
__builtin_parity	unsigned int	1 si x es par, 0 si es impar.
__builtin_XXXXXXll	unsigned ll	= pero para long long's.

2. Estructuras

2.1. Prefix table

```cpp
template<typename T>
class PTable {
    vec<T> pt;
    T opr (T a, T b) { return a + b; }
    T inv (T a, T b) { return a - b; }
    public:
    PTable (vec<T>& a) {
        pt.resize(a.size());
        rep(i,a.size()) pt[i] = !i ? a[i] : opr(pt[i-1], a[i]);
    }
    T q (int i, int j) { return (--i < 0) ? pt[j] : inv(pt[j], pt[i]); }
};
```

2.2. Sparse table

Operacion asociativa *idempotente*.

```cpp
template<typename T>
class STable {
    vec<vec<T>> st;
    T op (T a, T b) { return min(a,b); }
    public:
    STable (vec<T>& a) {
        st.resize(20, vec<T>(a.size()));
        st[0] = a; scn(w,1,19) scn(i,0,a.size()-(1<<w))
            st[w][i] = op(st[w-1][i], st[w-1][i+(1<<(w-1))]);
    }
    T q (int i, int j) {
        int w = log2fl(j - i + 1);
        return op(st[w][i], st[w][j - (1 << w) + 1]);
    }
};
```

2.3. Segment tree

```cpp
const int MAXN = 1<<20; // ~1e6

long long ST[2*MAXN];

long long querie(int nodo, int left, int right, int a, int b) {
    if (left >= b || right <= a) return 0;
    if (left >= a && right <= b) return ST[nodo];
    int m = (left + right) / 2;
    long long q1 = querie(nodo*2,left,m,a,b);
    long long q2 = querie(nodo*2+1,m,right,a,b);
    return q1 + q2;
}
void update(int p, long long val) {
    ST[p] = val;
    for (p = p/2; p > 0; p /= 2) ST[p] = op(ST[p*2], ST[p*2+1]);
}
```

2.4. Disjiont Set Union

```cpp
struct DSU {
    vec<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); }
};
```

3. Algos

3.1. Búsqueda Binaria (discreta)

```cpp
void dbisect (i64& i, i64& j, bool (*pred)(i64)) {
    while (!(i + 1 == j)) {
        i64 m = i + (j - i) / 2;
        pred(m) ? j = m : i = m;
    }
}
pair<bool, i64> dfirsttrue (i64 i, i64 j, bool (*pred)(i64)) {
    dbisect(i, j, pred);
    if (pred(i)) return mp(true, i);
    if (pred(j)) return mp(true, j);
    return mp(false, -1L);
}
pair<bool, i64> dlastfalse (i64 i, i64 j, bool (*pred)(i64)) {
    dbisect(i, j, pred);
    if (!pred(j)) return mp(true, j);
    if (!pred(i)) return mp(true, i);
    return mp(false, -1L);
}
```

4. Strings

5. Geometria

6. Math

6.1. Identidades

$\sum_{i=0}^{n} \binom{n}{i} = 2^n$
$\sum_{i=0}^{n} i\binom{n}{i} = n * 2^{n-1}$
$\sum_{i=m}^{n} i = \frac{n(n+1)}{2} - \frac{m(m-1)}{2} = \frac{(n+1-m)(n+m)}{2}$
$\sum_{i=0}^{n} i = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
$\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$
$\sum_{i=0}^{n} i(i-1) = \frac{8}{6}(\frac{n}{2})(\frac{n}{2}+1)(n+1)$ (doubles) → Sino ver caso impar y par
$\sum_{i=0}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4} = [\sum_{i=1}^{n} i]^2$
$\sum_{i=0}^{n} i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} = \frac{n^5}{5} + \frac{n^4}{2} + \frac{n^3}{3} - \frac{n}{30}$
$\sum_{i=0}^{n} i^p = \frac{(n+1)^{p+1}}{p+1} + \sum_{k=1}^{p} \frac{B_k}{p-k+1}\binom{p}{k}(n+1)^{p-k+1}$
$r = e - v + k + 1$
Teorema de Pick: (Area, puntos interiores y puntos en el borde)
$A = I + \frac{B}{2} - 1$

6.2. Criba

```
int es_compuesto[maxn]; vector<int> primos;
void criba (int n) {
    forf(i,2,n) if (!es_compuesto[i]) {
        primos.push_back(i); for (int j = 2; i*j < n; j++) es_compuesto[i*j] = 1;
    }
}
```

7. Grafos

7.1. Toposort

```
// Metodo 1: Flood fill con un reverse al final
// Facil
int topovisit[maxn]; vector<int> topoorder;
void topodfs (int u) {
    topovisit[u] = 1;
    for (int v : adj[u]) if (!topovisit[v]) {
        topodfs(v); topoorder.push_back(u);
    }
}
void toposort (void) {
    forf(u,0,n) if(!topovisit[u]) topodfs(u);
    reverse(topoorder.begin(), topoorder.end());
}
// Metodo 2: Basado en BFS;
// priority_queue permite priorizar entre vertices con mismo nivel topologico
int indegree[maxn]; vector<int> kahnsorder; void kahns () {
    priority_queue<int, vector<int>, greater<int>> pq;
    forf(u,0,n) for (int v : adj[u]) indegree[v]++;
    forf(u,0,n) if (!indegree[u]) pq.push(u);
    while (!pq.empty()) {
        int u = pq.top(); pq.pop(); kahnsorder.push_back(u);
        for (int v : adj[u]) { indegree[v]--; if (!indegree[v]) pq.push(v); }
    }
}
```

7.2. Componentes fuertemente conexas

```
const int maxn = int(5e5+1);
int n, m; vector<int> adj[maxn];

deque<int> kosaorder; int kosavisit[maxn];
void kosajaru1 (int u) { kosavisit[u] = 1; for (int v : adj[u]) if (!kosavisit[v]) kosajaru1(v); kosaorder.push_front(u); }
vector<int> tadj[maxn]; void kosajaru2 (void) { forf(u,0,n) for (int v : adj[u]) tadj[v].push_back(u); }
int kosaroot[maxn]; vector<vector<int>> kosacomp;
void kosajaru3 (int u, int r) { kosavisit[u] = 1; kosacomp.back().
```

```
    push_back(u); kosaroot[u] = r; for (int v : tadj[u]) if (!kosavisit[
    v]) kosajaru3(v, r); }
vector<int> adjscc[maxn]; int aristas[maxn]; void kosajaru (void) {
    forf(u,0,n) if (!kosavisit[u]) kosajaru1(u);
    kosajaru2(); memset(kosavisit, 0, maxn*sizeof(int));
    for (int u : kosaorder) if(!kosavisit[u]) { kosacomp.push_back({});
        kosajaru3(u, u); }
    memset(aristas, -1, sizeof(aristas));
    for (auto comp : kosacomp) for (int u : comp) for (int v : adj[u]) {
        int ru = kosaroot[u], rv = kosaroot[v];
        if (ru != rv && aristas[rv] != ru) aristas[rv] = ru, adjscc[ru].
            push_back(rv);
    }
}
```

7.3. Puentes y puntos de articulación

```
const int maxn = int(1e5+1);
int n, m; vector<int> adj[maxn];

vector<pair<int, int>> bridges; vector<int> arts;
int dfsdt[maxn], dfslow[maxn], dfsparent[maxn], dfstime, dfsroot,
    rootchildren, isart[maxn];
void artdfs (int u) { dfslow[u] =  dfsdt[u] = ++dfstime; for (int v :
    adj[u]) {
        if (!dfsdt[v]) {
            dfsparent[v] = u; if (u == dfsroot) ++rootchildren; artdfs(v
                );
            if (dfslow[v] >= dfsdt[u]) isart[u] = 1;
            if (dfslow[v] >  dfsdt[u]) bridges.push_back({u,v});
            dfslow[u] = min(dfslow[u], dfslow[v]);
        } else if (v != dfsparent[u]) dfslow[u] = min(dfslow[u], dfsdt[v
            ]);
    }
}
void getarts (void) { forf(u,0,n) dfsparent[u] = -1;
    forf(u,0,n) { if (!dfsdt[u]) { dfsroot = u; rootchildren = 0; artdfs
        (u); isart[dfsroot] = (rootchildren > 1); } }
    forf(u,0,n) if (isart[u]) arts.push_back(u);
}
```

8. Network Flow

9. Template

```
#include <bits/stdc++.h>
using namespace std;
using i64 = int64_t;
const int MAXN = 5e5;
const i64  INF = LLONG_MAX;
#define endl '\n'
#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 forall(it,v) for(auto it = v.begin(); it != v.end(); it++)
#define printall(v) forall(x,v){cout << *x << " ";} cout << endl
#define vec vector
#define pb push_back
#define mp make_pair
#define printpair(p) cout << "(" << p.fst << ", " << p.snd << ")" <<
    endl
#define fst first
#define snd second
#define log2fl(x) (x ? 63 - __builtin_clzll(x) : -1)
typedef vec<vec<int>> adj;
typedef vec<vec<pair<int,i64> > > wadj;

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

    // :)

    return 0;
}
```

10. Ayudamemoria

Compilar y correr

```
// 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
```