Índice

1. STL

Input/Forward Iterators (Generic)

min_element(first, last, [comp])	returns iterator to the (first) minimum
accumulate(first, last, init, [bp])	returns the result of sequentially applying *binary* operations starting from init
count(first, last, val)	returns the repetitions of val (using $==$)

Random Access Iterators (Arrays)

lower_bound(first, last, val, [comp])	returns iterator to the first element greater or equal than val (higher_bound is strict) [1]
sort(first, last, [comp<])	sorts elements in ascending order in $O(n \log n)$

Vectors

assign(first, last)	fills the vector using the range given

Sets and Maps

lower_bound(val)	returns iterator to the first element greater or equal than val (higher_bound is strict)

generic: insert(need to specify pos in vectors), erase, clear, swap. vectors have push_back, pop_back. Queues have push, pop, back, front. Stacks have push, pop, top. vectors have 'front' and 'back'

partition, set_union, set_difference, set_intersection, search, nth_element(intro select, qsort, median of medians magic), is_permutation

[1] If the iterator is random-access, the operation on average is logarithmic, otherwise it is linear

2. Yapas

Funciones Grandes

	5	**10**	**15**	**20**	**25**	**30**
$\mathbf{2^n}$	32	10^3	$3*10^4$	10^6	$3*10^7$	10^9
$\mathbf{3^n}$	243	$5*10^4$	10^7	$3*10^9$	...	...
$\binom{\mathbf{n}}{\mathbf{n/2}}$	10	252	$5*10^4$	10^5	10^6	10^8
$\mathbf{n!}$	120	10^{12}	...	...	...	...

Límites

INT_MAX $= 2e9$ LLONG_MAX $= 9e18$ DBL_MAX $= 1e37$

Lambdas

```
auto f = [] (int a, int b) { return a * b; };
std::accumulate(v.begin(), v.end(), 1, f);
```

Repetidos

```
set<int> s( vec.begin(), vec.end() );
vec.assign( s.begin(), s.end() );
```

3. Data Structures

La Sparse Table no es dinámica porque tiene mucha redundancia. Alternativamente, esa redundancia es la que le permite devolver queries en O(1) con operaciones idempotentes.

State: Unverified

```
struct Rmq {
    vector< vector<int> > st;
    Rmq(vector<int>& src) {
        st = vector< vector<int> >(floor(log2(sz(src))+1), vector<int>(sz(src)));
        forn(i, sz(src)) st[0][i] = src[i];
        forn(i, sz(st)-1) forn(j, sz(src)-(1<<i)) {
            st[i+1][j] = min(st[i][j], st[i][j+(1<<i)]);
        }
    }
    int query(int i, int j) {
        int niv = floor(log2(j-i));
        return min(st[niv][i], st[niv][j-(1<<niv)]);
    }
};
```

Segment Tree con mínimo puede ser utilizado usado para encontrar la primera ocurrencia de un elemento que cumpla con cierto predicado

4. Strings

Bordes

Un *borde* es un substring **estricto** que es prefijo y sufijo a la vez.

La aplicación será vista más tarde. A continuación se encuentra el desarrollo que nos va a permitir calcular bordes eficientemente.

Para todo string s y borde no trivial b, $b[1..n)$ es borde de $s[1..n)$. Luego, existe algun borde de $s[1..n)$ que puede extenderse a b. Por contrarecíproco, si no existe ninguno, b no es borde de s.

¿Cuándo es que un borde no máximo puede extenderse a borde del siguiente?

Ejemplos:

a	b	r	a	c	a	d	a	b	r	a
0	0	0	1	0	1	0	1	2	2	3

b	a	d	d	b	a	d	b	a
0	0	0	0	1	2	3	1	2

Esto da la pauta de como conseguir todos los bordes de s con DP, iterando sobre los bordes de $s[1..n)$ y verificando si pueden extenderse o no. Sin embargo, para cada prefijo tendría que calcular todos los bordes, y tendriamos que guardar $\mathbb{O}(n^2)$ bordes. Necesitamos una mejor forma de codificar nuestra información. Para esto vamos a aprovechar la relación que tienen los bordes de un string:

Lema. *Si A es borde de B y B es borde de C, entonces A es borde de C.*

Lema. *Dado un par de bordes de C, el menor es borde del mayor.*

Corolario. *Si $|A| < |B|$ y B es borde de C, entonces A borde de B si y solo si A borde de C*

Corolario. *Si M es el borde máximo de S, entonces para todo P distinto de S, P borde de S si y solo si P borde de M*

Ejemplo: malu*ma*luma $\rightarrow$ **ma**luma $\rightarrow$ ma

Lo que conseguimos con esto es darnos cuenta de que para calcular todos los bordes de un string basta con calcular el máximo borde de todos sus prefijos. Calculando un array con esta información, todos los bordes de un string s serían $mb[mb[...[mb[|s|]]]...]]$.

I mean, is that all there is?

5. Math

$$a^2 + bx + c = 0 \iff x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

6. Tips

Estrategias Generales

- **Simplificación:** Resolver una versión simplificada del problema. Suele ser más facil, y dar pauta de en donde se encuentra la dificultad en el problema original. También puede servir para encontrar una manera de construir la solución.
- **Fuerza Bruta:** Resolver el problema de forma pavota. Está bueno para tener ya una resolución posible, para poner mejor en contexto que tan óptima tenga que ser la respuesta.

Geometria

- Fijar grados de libertad de ser posible (desplazamiento, rotación)

C++

- std::find para sets es O(n), lo que es O(log n) es std::set::find
- El operador [] para maps crea un elemento si no lo encuentra, mejor usar find
- Módulo de negativos da negativo, ojo con eso