Binary Lifting

Sebastián Mestre[1]

[1]Universidad Nacional de Rosario
Don Gato

Training Camp 2025

¡Gracias sponsors!

Organizador

Diamond Plus

¡Gracias sponsors!

Platinum

FOLDER IT

INTERNATIONAL SOFTWARE COMPANY

Gold

NeuralSoft

JS JERÁRQUICOS

¡Gracias sponsors!

Aliado

$$x^n \pmod{m}$$

$$x \leq 10^9 \quad n \leq 10^{18} \quad m \leq 10^9$$

$$x^n = \begin{cases} x & \text{si } n = 1 \\ x \cdot x^{(n-1)} & \text{si } n > 0 \end{cases}$$

COSTO O(N)

$$* \quad x^{ab} = (x^a)^b$$

en particular:

- $x^n = x^{2n'} = (x^2)^{n'}$ donde $n = 2n'$ (n es par)

$$x^n = \begin{cases} x & \text{si } n = 1 \\ (x^2)^{n/2} & \text{si } n \text{ par, } n>0 \\ x \cdot x^{(n-1)} & \text{si } n \text{ impar} \end{cases}$$

$$x^n = \begin{cases} x & \text{si } n = 1 \\ (x^2)^{n/2} & \text{si } n \text{ par, } n>0 \\ x \cdot x^{(n-1)} = x \cdot (x^2)^{(n-1)/2} & \text{si } n \text{ impar} \end{cases}$$

COSTO O(log N)

Binary Lifting

```
int m;
int norm(int x) { return x - m * (x >= m); }
int add(int x, int y) { return norm(x + y); }
int sub(int x, int y) { return add(x, m - y); }
int mul(int x, int y) { return int(ll(x) * y % m); }

int pot(int x, int n) {
    if (n == 1) return x;
    if (n % 2 == 0) return pot(mul(x, x), n / 2);
    return mul(x, pot(x, n - 1));
}
```

Binary Lifting

```
int pot(int x, int n) {
    if (n == 1) return x;
    if (n % 2 == 0) return pot(x * x, n / 2);
    return x * pot(x, n - 1);
}
```

Análisis

$$
\begin{aligned}
& pot(x^1, 11) \\
= \; & x^1 * pot(x^1, 10) \\
= \; & x^1 * pot(x^2, 5) \\
= \; & x^1 * x^2 * pot(x^2, 4) \\
= \; & x^1 * x^2 * pot(x^4, 2) \\
= \; & x^1 * x^2 * pot(x^8, 1) \\
= \; & x^1 * x^2 * x^8
\end{aligned}
$$

$$x^{a+b} = x^a \cdot x^b$$

$$x^{11} = x^{8+2+1} = x^8 \cdot x^2 \cdot x^1$$

$$x^1$$

$$\rightarrow (x^1)^2 = x^2$$

$$\rightarrow (x^2)^2 = x^4$$

$$\rightarrow (x^4)^2 = x^8$$

$$\vdots$$

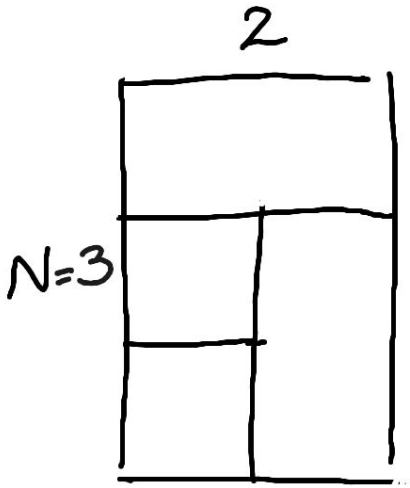
Análisis

```
int ps[logn]; // i -> pot(x, pot(2, i))
void init(int x) {
  ps[1] = x;
  forr(i, 2, logn) ps[i] = ps[i-1] * ps[i-1];
}

int pot(int n) {
  int y = 1;
  forn(i, logn) if (n&(1<<i)) y *= ps[i];
  return y;
}
```

Contar las formas de cubrir un rectángulo $N \times 2$ con rectángulos más chicos (mod 10^9+7)

$n \leq 10^6$



ANS = 34

doble

$d(n)$

simple

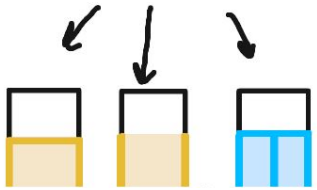
$s(n)$

doble

d(n)

simple

s(n)

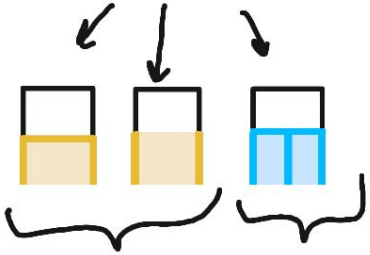


doble

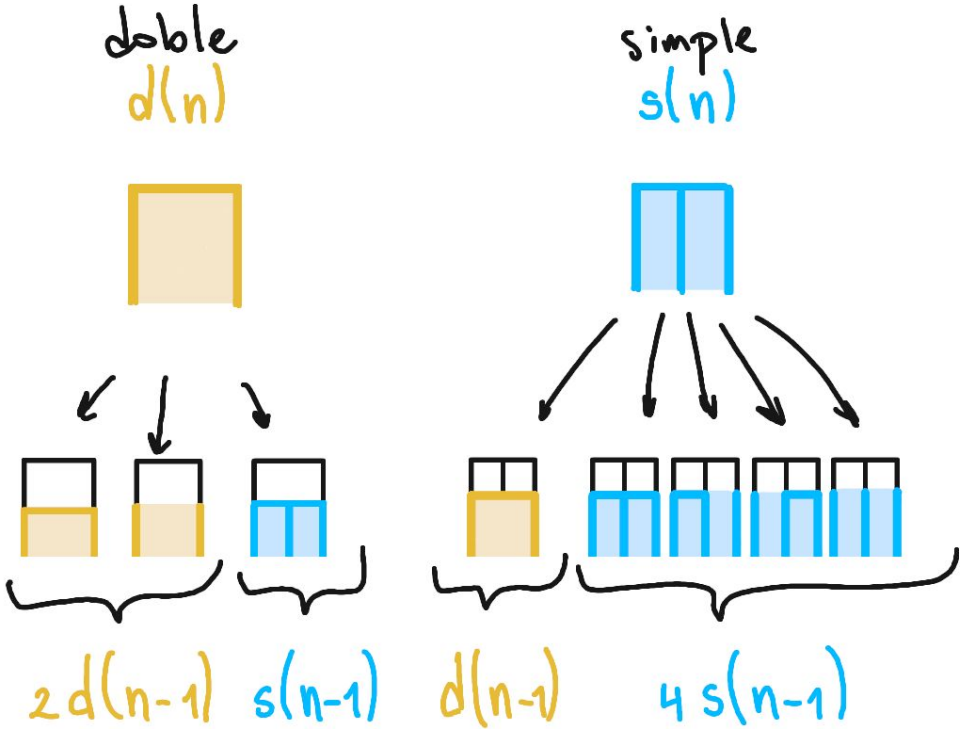
$d(n)$

simple

$s(n)$



$2d(n-1)$ $s(n-1)$


doble
d(n)
simple
s(n)
2 d(n-1)
s(n-1)
d(n-1)
4 s(n-1)

CSES - Counting Towers

```
int n;
cin >> n;
int simple = 1, doble = 1;
forn(i, n-1) {
    tie(simple, doble) = make_pair(
        4 * simple + doble,
        simple + 2 * doble);
}
cout << simple + doble << "\n";
```

$$\begin{pmatrix} s_n \\ d_n \end{pmatrix} = \begin{pmatrix} 4 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} s_{n-1} \\ d_{n-1} \end{pmatrix}$$

$$\begin{pmatrix} s_n \\ d_n \end{pmatrix} = \begin{pmatrix} 4 & 1 \\ 1 & 2 \end{pmatrix}^2 \begin{pmatrix} s_{n-2} \\ d_{n-2} \end{pmatrix}$$

$$\begin{pmatrix} s_n \\ d_n \end{pmatrix} = \begin{pmatrix} 4 & 1 \\ 1 & 2 \end{pmatrix}^3 \begin{pmatrix} s_{n-3} \\ d_{n-3} \end{pmatrix}$$

$$\begin{pmatrix} s_n \\ d_n \end{pmatrix} = \begin{pmatrix} 4 & 1 \\ 1 & 2 \end{pmatrix}^{n-1} \begin{pmatrix} s_1 \\ d_1 \end{pmatrix}$$

```
struct mat2x2 { int a, b,
                    c, d; };
mat2x2 operator*(mat2x2 x, mat2x2 y) {
  return { x.a*y.a + x.b*y.c, x.a*y.b + x.b*y.d,
           x.c*y.a + x.d*y.c, x.c*y.b + x.d*y.d };
}

struct vec2 { int a,
                  b; };
vec2 operator*(mat2x2 A, vec2 x) {
  return { A.a*x.a + A.b*c.b,
           A.a*x.c + A.b*c.d };
}
```

CSES - Counting Towers

```
int n;
cin >> n;
auto x = vec2{1,
              1};
auto A = mat2x2{4, 1,
                1, 2};
auto y = pot(A, n-1) * x;
cout << y.a + y.b << "\n";
```

Sea $(\star): A^2 \to A$ asociativa, y $n = n_1 + n_2 + \cdots + n_k$, donde $n_1, n_2, \ldots, n_k$ son potencias de 2 distintas.

Luego, para cada $x \in A$,

$$x^n = \underbrace{x \star x \star \cdots \star x}_{n \text{ veces}}$$

$$= x^{n_1} \star x^{n_2} \star \cdots \star x^{n_k}$$

Por lo tanto podemos aplicar binary lifting sobre cualquier $(A, \star)$ y el algoritmo aplica $(\star)$ $O(\log n)$ veces.

$$x^0 + x^1 + x^2 + \cdots + x^{n-1} \pmod{m}$$

$x \leq 10^9 \quad n \leq 10^{18} \quad m \leq 10^9$

(no primo !)

$$G(x,6) = x^0 + x^1 + x^2 + x^3 + x^4 + x^5$$

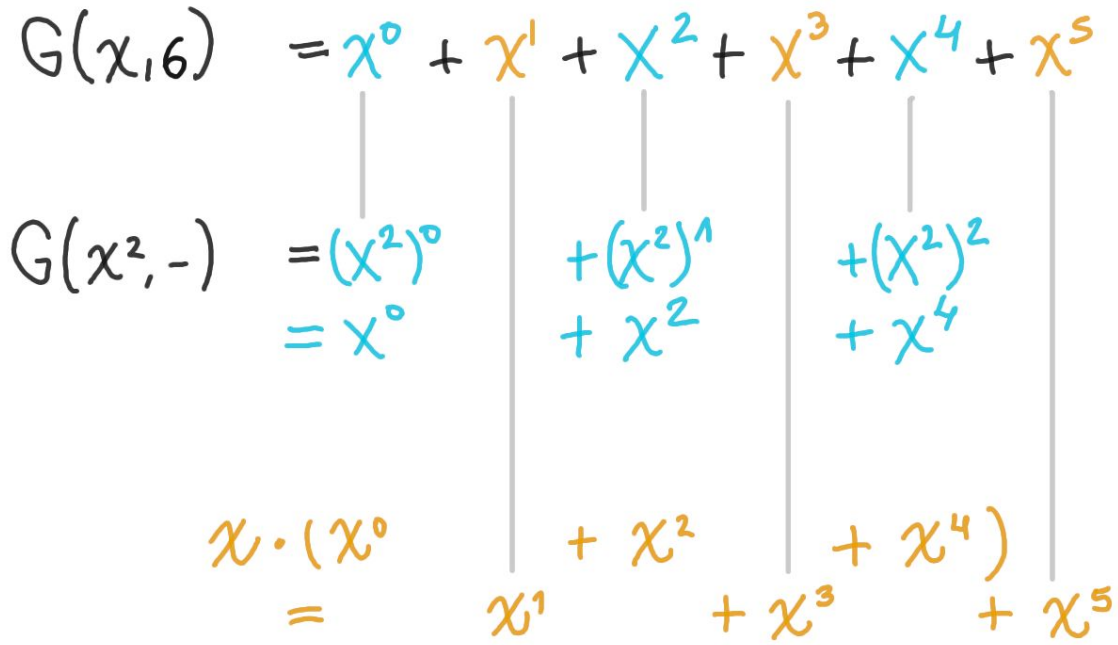
$$G(x,6) = x^0 + x^1 + x^2 + x^3 + x^4 + x^5$$

$$G(x^2,-) = (x^2)^0 + (x^2)^1 + (x^2)^2$$
$$= x^0 + x^2 + x^4$$

$$G(x,6) = x^0 + x^1 + x^2 + x^3 + x^4 + x^5$$

$$G(x^2,-) = (x^2)^0 + (x^2)^1 + (x^2)^2$$

$$= x^0 + x^2 + x^4$$


G(x,6) = x^0 + x^1 + x^2 + x^3 + x^4 + x^5
G(x^2,-) = (x^2)^0 + (x^2)^1 + (x^2)^2
= x^0 + x^2 + x^4
x · (x^0 + x^2 + x^4)
= x^1 + x^3 + x^5

$$G(x,n) = \begin{cases} x^0 = 1 & \text{si } n=1 \\ 1 + x \cdot G(x, n-1) & \text{si } n \text{ impar, } n>1 \\ (1+x) \cdot G(x^2, n/2) & \text{si } n \text{ par} \end{cases}$$

Suma geométrica

```
int geom(int x, int n) {
  if (n == 1) return 1;
  if (n % 2 == 0) return (1 + x) * geom(x * x, n / 2));
  return 1 + x * geom(x, n - 1);
}
```

Suma geométrica

```
int geom(int x, int n) {
  int r = 0;
  forn(i, n) r = x * r + 1;
  return r;
}
```

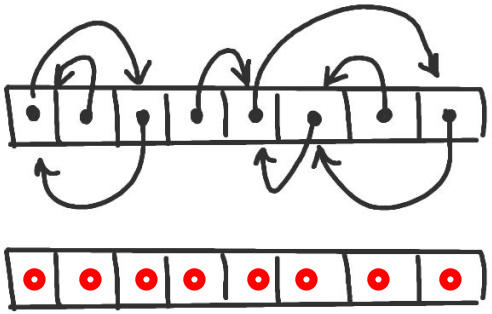
$$(x \mapsto ax+b) \circ (x \mapsto cx+d) = (x \mapsto acx + ad + b)$$

$$(x \mapsto ax+b)^3 = x \mapsto a^3x + \underbrace{a^2b + ab + b}_{b \cdot G(a,3)}$$

Suma Geométrica 2: Electric Boogaloo

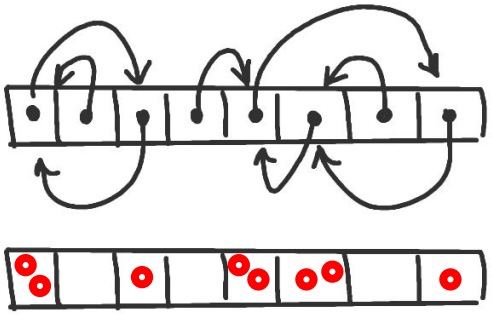
```
struct fn { int a, b; };
fn comp(fn f, fn g) {
  return { f.a * g.a, f.a * g.b + f.b };
}
int geom(int x, int n) {
  return pot(fn{x, 1}, n).b;
}
```

Cha-cha-cha



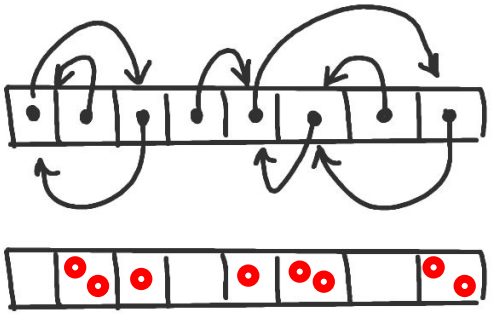
$n \leq 10^6$ $k \leq 10^{18}$

Cha-cha-cha

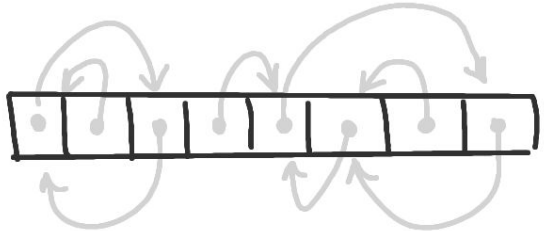


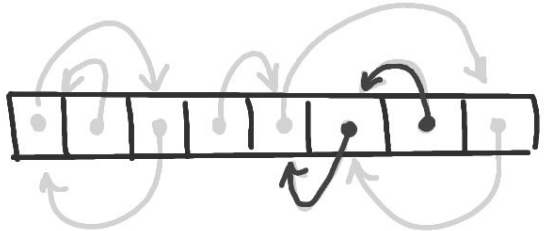
$n \leq 10^6$ $k \leq 10^{18}$

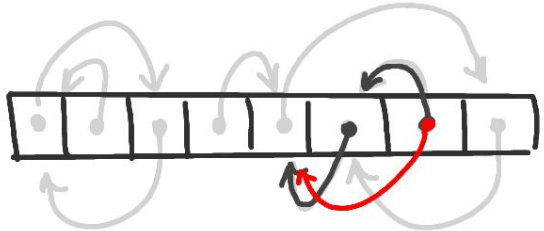
Cha-cha-cha

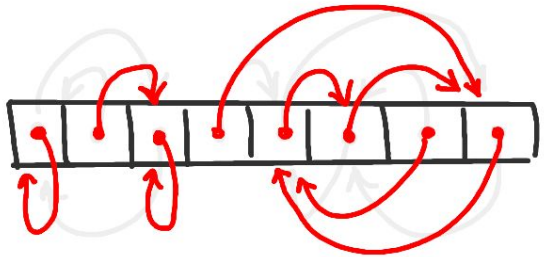


$n \leq 10^6$ $k \leq 10^{18}$



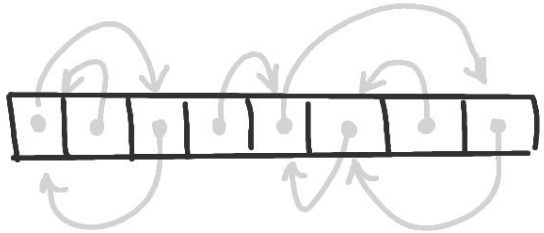


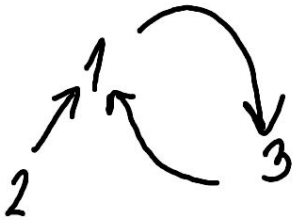


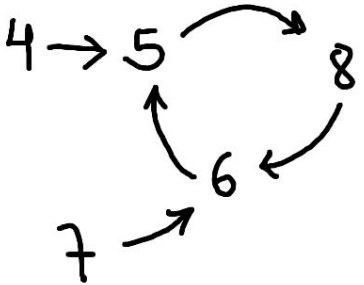


```
using baile = vector<int>;
baile operator*(baile f, baile g) {
  baile h(f.size());
  forn(x, f.size()) h[x] = f[g[x]];
  return h;
}
```

```
int n; long long k; cin >> n >> k;
baile a(n); forn(i, n) cin >> a[i];
baile b = pot(a, k);
vector<int> cant(n, 0);
forn(i, n) cant[b[i]]++;
forn(i, n) cout << cant[i] << " \n"[i==n-1];
```




1
2
3

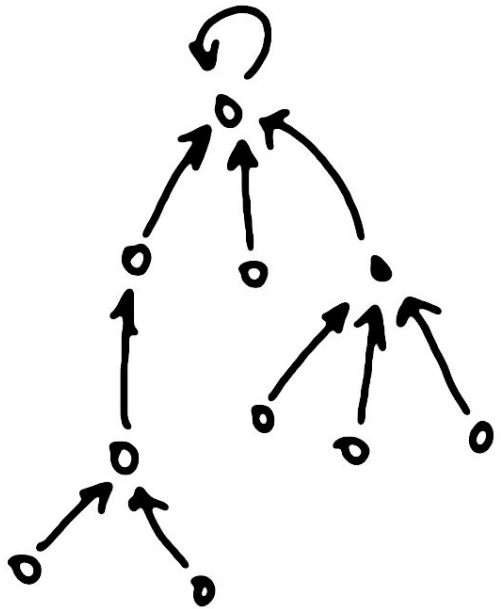

4
5
8
6
7

Dado un arbol T, definimos:

- $f(u)$ = padre de u en T
- $f^k(u)$ = k-ésimo ancestro de u en T

Responder Q consultas
$f^{k_i}(u_i)$ para $i = 1 \dots Q$

$Q \leq 10^6 \quad |T| \leq 10^6$



$$f^{11}(u) = f^{8+2+1}(u) = f^{8}(f^{2}(f^{1}(u)))$$

Idea: precalcular $f^{1}, f^{2}, f^{4}, \ldots, f^{2^{\lceil \lg |T| \rceil}}$

evaluar $f^{K_i}(u_i)$ "de a saltos"

Obligatory Ancestor Query Slide

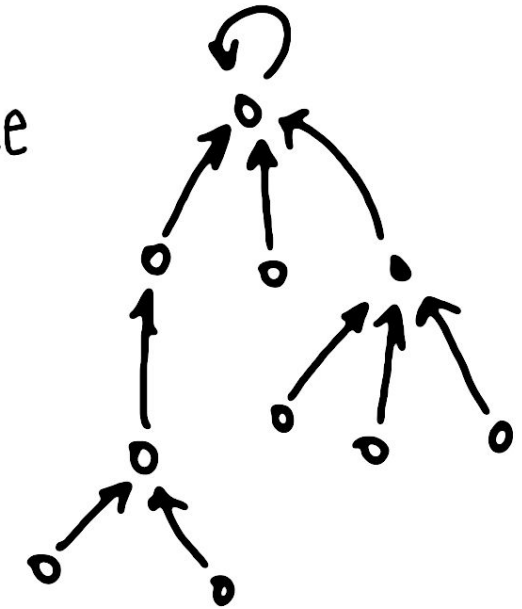
```
using arbol = vector<int>;
arbol ps[logn];
...
int eval_iter(int k, int u) {
  forn(i, logn) if (k&(1<<i)) u = ps[i][u];
  return u;
}
```

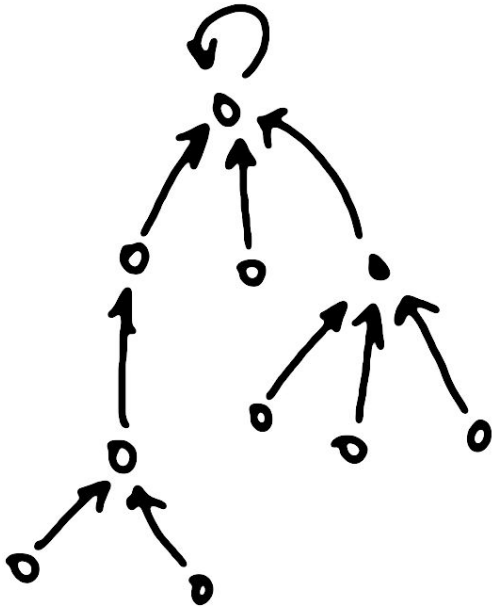
Dado un arbol T, definimos:

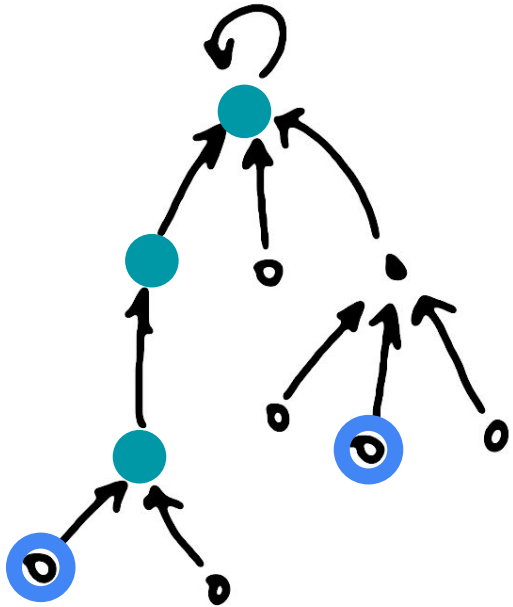
- $LCA(u,v)$ = nodo más profundo que es ancestro de u y v

Responder Q consultas $LCA(u_i, v_i)$ para $i = 1 \dots Q$

$Q \leq 10^6 \quad |T| \leq 10^6$







Obligatory LCA Slide

```
int first_true(int u, auto&& check) {
  if (check(u)) return u;
  dforn(i, logn) if (!check(ps[i][u])) u = ps[i][u];
  return ps[0][u];
}

bool ancestro(int u, int v) {
  return tin[u] <= tin[v] && tout[v] <= tout[u];
}

int lca(int u, int v) {
  return first_true(u, [&](int w) { return ancestro(w, v); });
}
```

Dado el arreglo $a_1, a_2, \cdots, a_n$

$$f(i, x) = (i+1, x+a_i)$$

$$f^k(i, x) = (i+k, x+a_i + a_{i+1} + \cdots + a_{i+k-1})$$

$$f^{r-l}(l,0) = (r,\ a_l + \cdots + a_{r-1})$$

$$f^1(i,0) = (i+1,\ a_i)$$
$$f^2(i,0) = (i+2,\ a_i + a_{i+1})$$
$$f^4(i,0) = (i+4,\ a_i + \cdots + a_{i+3})$$
$$\vdots$$

Sparse Table: trivialized

```
struct fn { int di; vector<int> a; };
fn comp(fn f, fn g) {
  fn h{f.di + g.di, vector<int>(g.a.size()-f.di)};
  forn(i, g.a.size()-f.di) h.a[i] = g.a[i] + f.a[i + g.di];
  return h;
}
pair<int,int> eval(fn f, int i, int x) {
  return {i+di, x+a[i]};
}
int range_sum(int l, int r) {
  return eval_iter(l, 0, r-l).second;
}
```

Extra

- truquito para detectar ciclos en grafo funcional
- funciones afines en espacios vectoriales o cuerpos (e.g. suma geométrica de matrices, hashes de strings repetidas)
- queries de suma/max/gcd en caminos en árboles/grafo funcional
- funciones biyectivas / permutaciones (charla de Mateo, mañana)
- en gral. composición de funciones "lindas"
- concatenar polinomios / hashes de strings repetidas
- multiplicar polinomios / convolución
- en gral. potencias en cualquier semigrupo