

# Humuhumunukunukuapua'a

## UFMG

Bruno Monteiro, Emanuel Silva e Bernardo Amorim

28 de Janeiro de 2023

## Índice

<b>1 DP</b>	<b>4</b>	2.8 Coloracao de Grafo de Intervalo . . . . .	12
1.1 Convex Hull Trick (Rafael) . . . . .	4	2.9 Conectividade Dinamica . . . . .	13
1.2 Convex Hull Trick Dinamico . . . . .	5	2.10 Conectividade Dinamica 2 . . . . .	13
1.3 Divide and Conquer DP . . . . .	5	2.11 Conj. Indep. Maximo com Peso em Grafo de Intervalo . . . . .	15
1.4 Longest Common Subsequence . . . . .	6	2.12 Distancia maxima entre dois pontos . . . . .	15
1.5 Mochila . . . . .	7	2.13 Distinct Range Query . . . . .	16
1.6 SOS DP . . . . .	7	2.14 Distinct Range Query com Update . . . . .	16
1.7 Subset sum . . . . .	7	2.15 Dominator Points . . . . .	17
<b>2 Problemas</b>	<b>8</b>	2.16 DP de Dominacao 3D . . . . .	18
2.1 Algoritmo Hungaro . . . . .	8	2.17 Gray Code . . . . .	18
2.2 Algoritmo MO - queries em caminhos de arvore . . . . .	8	2.18 Half-plane intersection . . . . .	19
2.3 Angle Range Intersection . . . . .	9	2.19 Heap Sort . . . . .	19
2.4 Area da Uniao de Retangulos . . . . .	10	2.20 Inversion Count . . . . .	19
2.5 Area Maxima de Histograma . . . . .	11	2.21 LIS - Longest Increasing Subsequence . . . . .	20
2.6 Binomial modular . . . . .	11	2.22 LIS2 - Longest Increasing Subsequence . . . . .	20
2.7 Closest pair of points . . . . .	12	2.23 Minimum Enclosing Circle . . . . .	20
		2.24 Minkowski Sum . . . . .	21
		2.25 MO - DSU . . . . .	21

2.26	Mo - numero de distintos em range . . . . .	22	<b>4 Matematica</b>	<b>38</b>
2.27	Palindromic Factorization . . . . .	23	4.1	2-SAT . . . . . 38
2.28	Parsing de Expressao . . . . .	23	4.2	Algoritmo de Euclides estendido . . . . . 39
2.29	RMQ com Divide and Conquer . . . . .	24	4.3	Avaliacao de Interpolacao . . . . . 39
2.30	Segment Intersection . . . . .	24	4.4	Berlekamp-Massey . . . . . 40
2.31	Sequencia de de Bruijn . . . . .	25	4.5	Binomial Distribution . . . . . 40
2.32	Shortest Addition Chain . . . . .	25	4.6	Convolucao de GCD / LCM . . . . . 41
2.33	Simple Polygon . . . . .	26	4.7	Coprime Basis . . . . . 41
2.34	Sweep Direction . . . . .	26	4.8	Crivo de Eratosthenes . . . . . 42
2.35	Triangulacao de Delaunay . . . . .	27	4.9	Deteccao de ciclo - Tortoise and Hare . . . . . 43
2.36	Triangulos em Grafos . . . . .	28	4.10	Division Trick . . . . . 44
<b>3</b>	<b>Strings</b>	<b>29</b>	4.11	Eliminacao Gaussiana . . . . . 44
3.1	Aho-corasick . . . . .	29	4.12	Eliminacao Gaussiana Z2 . . . . . 44
3.2	Algoritmo Z . . . . .	29	4.13	Equacao Diofantina Linear . . . . . 45
3.3	Automato de Sufixo . . . . .	29	4.14	Exponenciacao rapida . . . . . 45
3.4	eertree . . . . .	30	4.15	Fast Walsh Hadamard Transform . . . . . 46
3.5	KMP . . . . .	31	4.16	FFT . . . . . 46
3.6	Manacher . . . . .	31	4.17	Integracao Numerica - Metodo de Simpson 3/8 . . . . . 47
3.7	Min/max suffix/cyclic shift . . . . .	32	4.18	Inverso Modular . . . . . 47
3.8	String Hashing . . . . .	32	4.19	Karatsuba . . . . . 48
3.9	String Hashing - modulo $2^{61} - 1$ . . . . .	33	4.20	Logaritmo Discreto . . . . . 48
3.10	Suffix Array - $O(n \log n)$ . . . . .	33	4.21	Miller-Rabin . . . . . 49
3.11	Suffix Array - $O(n)$ . . . . .	33	4.22	Pollard's Rho Alg . . . . . 49
3.12	Suffix Array Dinamico . . . . .	36	4.23	Produto de dois long long mod m . . . . . 50
3.13	Trie . . . . .	38	4.24	Simplex . . . . . 50
			4.25	Teorema Chines do Resto . . . . . 51

4.26	Totiente . . . . .	51	6.15	SegTree . . . . .	77
<b>5</b>	<b>Primitivas</b>	<b>51</b>	6.16	SegTree 2D Iterativa . . . . .	78
5.1	Aritmetica Modular . . . . .	51	6.17	SegTree Beats . . . . .	78
5.2	Big Integer . . . . .	52	6.18	SegTree Colorida . . . . .	80
5.3	Matroid . . . . .	55	6.19	SegTree Esparsa - Lazy . . . . .	81
5.4	Primitivas de fracao . . . . .	57	6.20	SegTree Esparsa - $O(q)$ memoria . . . . .	82
5.5	Primitivas de matriz - exponenciacao . . . . .	57	6.21	SegTree Iterativa . . . . .	83
5.6	Primitivas Geometricas . . . . .	58	6.22	SegTree Iterativa com Lazy Propagation . . . . .	83
5.7	Primitivas Geometricas 3D . . . . .	63	6.23	SegTree PA . . . . .	84
5.8	Primitivas Geometricas Inteiras . . . . .	65	6.24	SegTree Persistente . . . . .	85
<b>6</b>	<b>Estruturas</b>	<b>68</b>	6.25	Sparse Table . . . . .	86
6.1	BIT . . . . .	68	6.26	Sparse Table Disjunta . . . . .	86
6.2	BIT 2D . . . . .	68	6.27	Splay Tree . . . . .	86
6.3	BIT com update em range . . . . .	69	6.28	Splay Tree Implicita . . . . .	88
6.4	BIT-Sort Tree . . . . .	69	6.29	Split-Merge Set . . . . .	90
6.5	DSU . . . . .	70	6.30	SQRT Tree . . . . .	91
6.6	Li-Chao Tree . . . . .	71	6.31	Treap . . . . .	92
6.7	MergeSort Tree . . . . .	71	6.32	Treap Implicita . . . . .	93
6.8	Min queue - deque . . . . .	73	6.33	Treap Persistent Implicita . . . . .	94
6.9	Min queue - stack . . . . .	73	6.34	Wavelet Tree . . . . .	95
6.10	Order Statistic Set . . . . .	73	<b>7</b>	<b>Grafos</b>	<b>96</b>
6.11	Priority Queue DS . . . . .	74	7.1	AGM Direcionada . . . . .	96
6.12	Range color . . . . .	75	7.2	Articulation Points . . . . .	97
6.13	RMQ $\langle O(n), O(1) \rangle$ - min queue . . . . .	75	7.3	Bellman-Ford . . . . .	97
6.14	SegTreap . . . . .	75	7.4	Block-Cut Tree . . . . .	97

7.5	Blossom - matching maximo em grafo geral . . . . .	98	7.31	Max flow com lower bound nas arestas . . . . .	117
7.6	Centro de arvore . . . . .	99	7.32	MinCostMaxFlow . . . . .	118
7.7	Centroid . . . . .	100	7.33	Prufer code . . . . .	119
7.8	Centroid decomposition . . . . .	100	7.34	Sack (DSU em arvores) . . . . .	120
7.9	Centroid Tree . . . . .	100	7.35	Stable Marriage . . . . .	120
7.10	Dijkstra . . . . .	101	7.36	Tarjan para SCC . . . . .	121
7.11	Dinitz . . . . .	101	7.37	Topological Sort . . . . .	121
7.12	Dominator Tree - Kawakami . . . . .	102	7.38	Vertex cover . . . . .	122
7.13	Euler Path / Euler Cycle . . . . .	103	7.39	Virtual Tree . . . . .	122
7.14	Euler Tour Tree . . . . .	104	<b>8</b>	<b>Extra</b>	<b>123</b>
7.15	Floyd-Warshall . . . . .	106	8.1	gethash.sh . . . . .	123
7.16	Functional Graph . . . . .	106	8.2	hash.sh . . . . .	123
7.17	Heavy-Light Decomposition - aresta . . . . .	107	8.3	makefile . . . . .	123
7.18	Heavy-Light Decomposition - vertice . . . . .	108	8.4	fastIO.cpp . . . . .	123
7.19	Heavy-Light Decomposition sem Update . . . . .	109	8.5	vimrc . . . . .	124
7.20	Isomorfismo de arvores . . . . .	109	8.6	stress.sh . . . . .	124
7.21	Kosaraju . . . . .	110	8.7	rand.cpp . . . . .	124
7.22	Kruskal . . . . .	110	8.8	timer.cpp . . . . .	124
7.23	Kuhn . . . . .	111	8.9	debug.cpp . . . . .	124
7.24	LCA com binary lifting . . . . .	111	8.10	template.cpp . . . . .	124
7.25	LCA com HLD . . . . .	112	<b>1</b>	<b>DP</b>	
7.26	LCA com RMQ . . . . .	113	<b>1.1</b>	<b>Convex Hull Trick (Rafael)</b>	
7.27	Line Tree . . . . .	114		// adds tem que serem feitos em ordem de slope	
7.28	Link-cut Tree . . . . .	114		// queries tem que ser feitas em ordem de x	
7.29	Link-cut Tree - aresta . . . . .	115			
7.30	Link-cut Tree - vertice . . . . .	116			

```
//
// linear
// 30323e

4b 4b struct CHT {
af 94     int it;
7f ac     vector<ll> a, b;
8c 45     CHT():it(0){}
fc 0b     ll eval(int i, ll x){
24 93         return a[i]*x + b[i];
6b cb     }
f1 63     bool useless(){
18 a2         int sz = a.size();
06 35         int r = sz-1, m = sz-2, l = sz-3;
22 d7         return (b[l] - b[r])*(a[m] - a[l]) <
a5 41             (b[l] - b[m])*(a[r] - a[l]);
ff cb     }
9d bf     void add(ll A, ll B){
c6 7f         a.push_back(A); b.push_back(B);
06 56         while (!a.empty()){
0b 23             if ((a.size() < 3) || !useless()) break;
a7 ec             a.erase(a.end() - 2);
89 56             b.erase(b.end() - 2);
2f cb         }
a3 cb     }
29 81     ll get(ll x){
9b d2         it = min(it, int(a.size()) - 1);
d7 46         while (it+1 < a.size()){
51 3c             if (eval(it+1, x) > eval(it, x)) it++;
99 f9             else break;
cf cb         }
37 42         return eval(it, x);
7f cb     }
30 21 };

```

## 1.2 Convex Hull Trick Dinamico

```
// para double, use LINF = 1/.0, div(a, b) = a/b
// update(x) atualiza o ponto de intersecao da reta x
// overlap(x) verifica se a reta x sobrepoe a proxima
// add(a, b) adiciona reta da forma ax + b
// query(x) computa maximo de ax + b para entre as retas
//
// O(log(n)) amortizado por insercao
// O(log(n)) por query
// 978376

```

```
72 72 struct Line {
90 07     mutable ll a, b, p;
2c 8e     bool operator<(const Line& o) const { return a < o.a; }
58 ab     bool operator<(ll x) const { return p < x; }
46 21 };

99 32 struct dynamic_hull : multiset<Line, less<>> {
8d 33     ll div(ll a, ll b) {
ef a2         return a / b - ((a ^ b) < 0 and a % b);
0f cb     }

b1 bb     void update(iterator x) {
c8 b2         if (next(x) == end()) x->p = LINF;
6a 77         else if (x->a == next(x)->a) x->p = x->b >= next(x)->b ?
LINF : -LINF;
4b 42         else x->p = div(next(x)->b - x->b, x->a - next(x)->a);
5d cb     }

e9 71     bool overlap(iterator x) {
14 f1         update(x);
5c cf         if (next(x) == end()) return 0;
b4 a4         if (x->a == next(x)->a) return x->b >= next(x)->b;
2f d4         return x->p >= next(x)->p;
70 cb     }

03 17     void add(ll a, ll b) {
48 1c         auto x = insert({a, b, 0});
73 4a         while (overlap(x)) erase(next(x)), update(x);
da db         if (x != begin() and !overlap(prev(x))) x = prev(x),
update(x);
64 0f         while (x != begin() and overlap(prev(x)))
2f 4d             x = prev(x), erase(next(x)), update(x);
04 cb     }

ad 4a     ll query(ll x) {
64 22         assert(!empty());
f0 7d         auto l = *lower_bound(x);
0a ab         return l.a * x + l.b;
f5 cb     }
97 21 };

```

## 1.3 Divide and Conquer DP

```
// Particiona o array em k subarrays
// minimizando o somatorio das queries
//
// O(k n log n), assumindo quer query(l, r) eh O(1)

```

```

// 4efe6b

54 54 11 dp[MAX][2];

ab 94 void solve(int k, int l, int r, int lk, int rk) {
7c de     if (l > r) return;
80 10     int m = (l+r)/2, p = -1;
d8 d2     auto& ans = dp[m][k&1] = LINF;
18 6e     for (int i = max(m, lk); i <= rk; i++) {
e6 32         int at = dp[i+1][~k&1] + query(m, i);
bc 57         if (at < ans) ans = at, p = i;
a1 cb     }
79 1e     solve(k, l, m-1, lk, p), solve(k, m+1, r, p, rk);
b5 cb }

9b cf 11 DC(int n, int k) {
e8 32     dp[n][0] = dp[n][1] = 0;
02 f2     for (int i = 0; i < n; i++) dp[i][0] = LINF;
9b b7     for (int i = 1; i <= k; i++) solve(i, 0, n-i, 0, n-i);
4e 8e     return dp[0][k&1];
4e cb }

```

## 1.4 Longest Common Subsequence

```

// Computa a LCS entre dois arrays usando
// o algoritmo de Hirschberg para recuperar
//
// O(n*m), O(n+m) de memoria
// 337bb3

ea ea int lcs_s[MAX], lcs_t[MAX];
da a6 int dp[2][MAX];

// dp[0][j] = max lcs(s[li...ri], t[lj, lj+j])
0b d1 void dp_top(int li, int ri, int lj, int rj) {
73 d1     memset(dp[0], 0, (rj-lj+1)*sizeof(dp[0][0]));
67 75     for (int i = li; i <= ri; i++) {
e7 9a         for (int j = rj; j >= lj; j--)
7d 83             dp[0][j - lj] = max(dp[0][j - lj],
76 74                 (lcs_s[i] == lcs_t[j]) + (j > lj ? dp[0][j-1 - lj] :
0));
cc 04         for (int j = lj+1; j <= rj; j++)
d9 93             dp[0][j - lj] = max(dp[0][j - lj], dp[0][j-1 -lj]);
8b cb     }
87 cb }

// dp[1][j] = max lcs(s[li...ri], t[lj+j, rj])

```

```

c5 ca void dp_bottom(int li, int ri, int lj, int rj) {
34 0d     memset(dp[1], 0, (rj-lj+1)*sizeof(dp[1][0]));
4d 3a     for (int i = ri; i >= li; i--) {
c8 49         for (int j = lj; j <= rj; j++)
bc db             dp[1][j - lj] = max(dp[1][j - lj],
c5 4d                 (lcs_s[i] == lcs_t[j]) + (j < rj ? dp[1][j+1 - lj] :
0));
e7 6c         for (int j = rj-1; j >= lj; j--)
dc 76             dp[1][j - lj] = max(dp[1][j - lj], dp[1][j+1 - lj]);
5e cb     }
c5 cb }

d5 93 void solve(vector<int>& ans, int li, int ri, int lj, int rj) {
6c 2a     if (li == ri){
d7 49         for (int j = lj; j <= rj; j++)
57 f5             if (lcs_s[li] == lcs_t[j]){
59 a6                 ans.push_back(lcs_t[j]);
08 c2                 break;
e1 cb             }
fa 50             return;
29 cb         }
a1 53         if (lj == rj){
f9 75             for (int i = li; i <= ri; i++){
c1 88                 if (lcs_s[i] == lcs_t[lj]){
2a 53                     ans.push_back(lcs_s[i]);
2b c2                     break;
c1 cb                 }
c4 cb             }
04 50             return;
54 cb         }
24 a5         int mi = (li+ri)/2;
32 ad         dp_top(li, mi, lj, rj), dp_bottom(mi+1, ri, lj, rj);

5c d7         int j_ = 0, mx = -1;

aa ae         for (int j = lj-1; j <= rj; j++) {
56 da             int val = 0;
ad 2b             if (j >= lj) val += dp[0][j - lj];
33 b9             if (j < rj) val += dp[1][j+1 - lj];

0a ba                 if (val >= mx) mx = val, j_ = j;
a6 cb             }
fc 6f             if (mx == -1) return;
28 c2             solve(ans, li, mi, lj, j_), solve(ans, mi+1, ri, j_+1, rj);
67 cb }

2a 05 vector<int> lcs(const vector<int>& s, const vector<int>& t) {

```

```

35 95     for (int i = 0; i < s.size(); i++) lcs_s[i] = s[i];
3d 57     for (int i = 0; i < t.size(); i++) lcs_t[i] = t[i];
b9 da     vector<int> ans;
2a 59     solve(ans, 0, s.size()-1, 0, t.size()-1);
1b ba     return ans;
33 cb }

```

## 1.5 Mochila

```

// Resolve mochila, recuperando a resposta
//
// O(n * cap), O(n + cap) de memoria
// 400885

ad ad int v[MAX], w[MAX]; // valor e peso
d0 58 int dp[2][MAX_CAP];

// DP usando os itens [l, r], com capacidade = cap
ce 0d void get_dp(int x, int l, int r, int cap) {
77 f8     memset(dp[x], 0, (cap+1)*sizeof(dp[x][0]));
8f 57     for (int i = l; i <= r; i++) for (int j = cap; j >= 0; j--)
1c 3a         if (j - w[i] >= 0) dp[x][j] = max(dp[x][j], v[i] + dp[x][j
- w[i]]);
10 cb }

17 5a void solve(vector<int>& ans, int l, int r, int cap) {
a7 89     if (l == r) {
38 9f         if (w[l] <= cap) ans.push_back(l);
ea 50         return;
e8 cb     }
98 ee     int m = (l+r)/2;
88 28     get_dp(0, l, m, cap), get_dp(1, m+1, r, cap);
92 05     int left_cap = -1, opt = -INF;
38 c9     for (int j = 0; j <= cap; j++)
3b 2f         if (int at = dp[0][j] + dp[1][cap - j]; at > opt)
57 91             opt = at, left_cap = j;
fc da     solve(ans, l, m, left_cap), solve(ans, m+1, r, cap - left_cap);
cc cb }

be 0d vector<int> knapsack(int n, int cap) {
ce da     vector<int> ans;
bb 1e     solve(ans, 0, n-1, cap);
47 ba     return ans;
40 cb }

```

## 1.6 SOS DP

```

// O(n 2^n)

// soma de sub-conjunto
// bec381
e0 e0 vector<ll> sos_dp(vector<ll> f) {
98 6c     int N = __builtin_ctz(f.size());
d8 e5     assert((1<<N) == f.size());

5a 5a     for (int i = 0; i < N; i++) for (int mask = 0; mask < (1<<N);
mask++)
d9 79         if (mask>>i&1) f[mask] += f[mask^(1<<i)];
f2 ab     return f;
be cb }

// soma de super-conjunto
// dbd121
38 e0 vector<ll> sos_dp(vector<ll> f) {
2a 6c     int N = __builtin_ctz(f.size());
0e e5     assert((1<<N) == f.size());

b9 5a     for (int i = 0; i < N; i++) for (int mask = 0; mask < (1<<N);
mask++)
9f a3         if (~mask>>i&1) f[mask] += f[mask^(1<<i)];
56 ab     return f;
ff cb }

```

## 1.7 Subset sum

```

// Retorna max(x <= t tal que existe subset de w que soma x)
//
// O(n * max(w))
// O(max(w)) de memoria
// d888b0

ef ef int subset_sum(vector<int> w, int t) {
88 bb     int pref = 0, k = 0;
0d 41     while (k < w.size() and pref + w[k] <= t) pref += w[k++];
da 1e     if (k == w.size()) return pref;
77 44     int W = *max_element(w.begin(), w.end());
27 44     vector<int> last, dp(2*W, -1);
2f d7     dp[W - (t-pref)] = k;
da 54     for (int i = k; i < w.size(); i++) {
e5 28         last = dp;
8a 15         for (int x = 0; x < W; x++) dp[x+w[i]] = max(dp[x+w[i]],
last[x]);
ba 17         for (int x = 2*W - 1; x > W; x--)
56 30             for (int j = max(0, last[x]); j < dp[x]; j++)

```

```

c2 59         dp[x-w[j]] = max(dp[x-w[j]], j);
8c cb     }
59 2f     int ans = t;
eb 1c     while (dp[W - (t-ans)] < 0) ans--;
18 ba     return ans;
d8 cb }

```

## 2 Problemas

### 2.1 Algoritmo Hungaro

```

// Resolve o problema de assignment (matriz n x n)
// Colocar os valores da matriz em 'a' (pode < 0)
// assignment() retorna um par com o valor do
// assignment minimo, e a coluna escolhida por cada linha
//
// O(n^3)
// 64c53e

a6 a6 template<typename T> struct hungarian {
8e 1a     int n;
28 a0     vector<vector<T>> a;
a1 f3     vector<T> u, v;
21 5f     vector<int> p, way;
54 f1     T inf;

7c c3     hungarian(int n_) : n(n_), u(n+1), v(n+1), p(n+1), way(n+1) {
ab b2         a = vector<vector<T>>(n, vector<T>(n));
ef 1f         inf = numeric_limits<T>::max();
59 cb     }
6e d6     pair<T, vector<int>> assignment() {
9c 78         for (int i = 1; i <= n; i++) {
31 8c             p[0] = i;
a9 62             int j0 = 0;
03 ce             vector<T> minv(n+1, inf);
71 24             vector<int> used(n+1, 0);
57 01             do {
1f 47                 used[j0] = true;
42 d2                 int i0 = p[j0], j1 = -1;
91 7e                 T delta = inf;
f6 9a                 for (int j = 1; j <= n; j++) if (!used[j]) {
aa 7b                     T cur = a[i0-1][j-1] - u[i0] - v[j];
dc 9f                     if (cur < minv[j]) minv[j] = cur, way[j] = j0;
72 82                     if (minv[j] < delta) delta = minv[j], j1 = j;
3f cb                 }
a8 f6                 for (int j = 0; j <= n; j++)

```

```

43 2c             if (used[j]) u[p[j]] += delta, v[j] -= delta;
1a 6e             else minv[j] -= delta;
4b 6d             j0 = j1;
f2 23             } while (p[j0] != 0);
d0 01             do {
bf 4c                 int j1 = way[j0];
97 0d                 p[j0] = p[j1];
4a 6d                 j0 = j1;
cb ca             } while (j0);
5c cb         }
8e 30         vector<int> ans(n);
21 6d         for (int j = 1; j <= n; j++) ans[p[j]-1] = j-1;
36 da         return make_pair(-v[0], ans);
af cb     }
64 21 };

```

### 2.2 Algoritmo MO - queries em caminhos de arvore

```

// Problema que resolve: https://www.spoj.com/problems/COT2/
//
// Complexidade sendo c = O(update) e SQ = sqrt(n):
// O((n + q) * sqrt(n) * c)
// 395329

1b 1b const int MAX = 40010, SQ = 400;

de 04 vector<int> g[MAX];

b8 c5 namespace LCA { ... }

da 24 int in[MAX], out[MAX], vtx[2 * MAX];
5d 81 bool on[MAX];

3d 4c int dif, freq[MAX];
41 9e vector<int> w;

2c d9 void dfs(int v, int p, int &t) {
09 65     vtx[t] = v, in[v] = t++;
34 18     for (int u : g[v]) if (u != p) {
66 c5         dfs(u, v, t);
f8 cb     }
55 21     vtx[t] = v, out[v] = t++;
01 cb }

1e e5 void update(int p) { // faca alteracoes aqui
5d bb     int v = vtx[p];
91 0e     if (not on[v]) { // insere vtx v

```



```

e9 31         dif += (freq[w[v]] == 0);
05 b2         freq[w[v]]++;
4d cb     }
44 4e     else { // retira o vertice v
8f 0a         dif -= (freq[w[v]] == 1);
8c fd         freq[w[v]]--;
53 cb     }
aa 73     on[v] = not on[v];
9e cb }

ca a3 vector<tuple<int, int, int>> build_queries(const
    vector<pair<int, int>>& q) {
51 ea     LCA::build(0);
4e f7     vector<tuple<int, int, int>> ret;
ce aa     for (auto [l, r] : q){
a6 d2         if (in[r] < in[l]) swap(l, r);
68 6f         int p = LCA::lca(l, r);
b2 82         int init = (p == l) ? in[l] : out[l];
1b 07         ret.emplace_back(init, in[r], in[p]);
da cb     }
23 ed     return ret;
44 cb }

b3 f3 vector<int> mo_tree(const vector<pair<int, int>>& vq){
ea 6b     int t = 0;
77 da     dfs(0, -1, t);

c0 af     auto q = build_queries(vq);

f7 f4     vector<int> ord(q.size());
4d be     iota(ord.begin(), ord.end(), 0);
11 d0     sort(ord.begin(), ord.end(), [&] (int l, int r) {
d0 d8         int bl = get<0>(q[l]) / SQ, br = get<0>(q[r]) / SQ;
25 59         if (bl != br) return bl < br;
02 15         else if (bl % 2 == 1) return get<1>(q[l]) < get<1>(q[r]);
9f f1         else return get<1>(q[l]) > get<1>(q[r]);
c7 c0     });

96 80     memset(freq, 0, sizeof freq);
3f bf     dif = 0;

cf ff     vector<int> ret(q.size());
4f 3d     int l = 0, r = -1;
a3 8b     for (int i : ord) {
a3 3c         auto [ql, qr, qp] = q[i];
46 af         while (r < qr) update(++r);
64 d6         while (l > ql) update(--l);

```

```

e9 95         while (l < ql) update(l++);
e9 6a         while (r > qr) update(r--);

5f 3d         if (qp < l or qp > r) { // se LCA estah entre as pontas
90 74             update(qp);
45 2e             ret[i] = dif;
1b 74             update(qp);
16 cb         }
52 0f         else ret[i] = dif;
95 cb     }
93 ed     return ret;
39 cb }

```

## 2.3 Angle Range Intersection

```

// Computa intersecao de angulos
// Os angulos (arcos) precisam ter comprimento < pi
// (caso contrario a intersecao eh estranha)
//
// Tudo 0(1)
// 5e1c85

32 32 struct angle_range {
02 75     static constexpr ld ALL = 1e9, NIL = -1e9;
58 39     ld l, r;
4e c7     angle_range() : l(ALL), r(ALL) {}
81 89     angle_range(ld l_, ld r_) : l(l_), r(r_) { fix(l), fix(r); }

7d 4e     void fix(ld& theta) {
71 da         if (theta == ALL or theta == NIL) return;
3f 32         if (theta > 2*pi) theta -= 2*pi;
eb 86         if (theta < 0) theta += 2*pi;
3a cb     }
0b 2e     bool empty() { return l == NIL; }
da 93     bool contains(ld q) {
18 40         fix(q);
cd 4d         if (l == ALL) return true;
ee fe         if (l == NIL) return false;
f2 6a         if (l < r) return l < q and q < r;
85 07         return q > l or q < r;
12 cb     }
43 9c     friend angle_range operator &(angle_range p, angle_range q) {
b6 74         if (p.l == ALL or q.l == NIL) return q;
7d 20         if (q.l == ALL or p.l == NIL) return p;
de 7d         if (p.l > p.r and q.l > q.r) return {max(p.l, q.l) ,
            min(p.r, q.r)};
06 aa         if (q.l > q.r) swap(p.l, q.l), swap(p.r, q.r);

```

```

d9 8d      if (p.l > p.r) {
76 24          if (q.r > p.l) return {max(q.l, p.l) , q.r};
eb 6f          else if (q.l < p.r) return {q.l, min(q.r, p.r)};
2d 27          return {NIL, NIL};
f2 cb      }
93 5a      if (max(p.l, q.l) > min(p.r, q.r)) return {NIL, NIL};
3b bc      return {max(p.l, q.l), min(p.r, q.r)};
f8 cb      }
5e 21 };

```

## 2.4 Area da Uniao de Retangulos

```

// O(n log(n))
// 5d8d2f

```

```

aa aa namespace seg {
71 6b     pair<int, ll> seg[4*MAX];
5c b1     ll lazy[4*MAX], *v;
a4 1a     int n;

10 e0     pair<int, ll> merge(pair<int, ll> l, pair<int, ll> r){
07 71         if (l.second == r.second) return {l.first+r.first,
1. second};
d7 53         else if (l.second < r.second) return l;
0e aa         else return r;
c1 cb     }

13 6f     pair<int, ll> build(int p=1, int l=0, int r=n-1) {
26 3c         lazy[p] = 0;
37 bf         if (l == r) return seg[p] = {1, v[l]};
d4 ee         int m = (l+r)/2;
3e 43         return seg[p] = merge(build(2*p, l, m), build(2*p+1, m+1,
r));
12 cb     }
b4 d9     void build(int n2, ll* v2) {
08 68         n = n2, v = v2;
de 6f         build();
23 cb     }
e4 ce     void prop(int p, int l, int r) {
65 20         seg[p].second += lazy[p];
88 2c         if (l != r) lazy[2*p] += lazy[p], lazy[2*p+1] += lazy[p];
24 3c         lazy[p] = 0;
72 cb     }
ea 69     pair<int, ll> query(int a, int b, int p=1, int l=0, int r=n-1)
{
39 6b         prop(p, l, r);
cc 52         if (a <= l and r <= b) return seg[p];

```

```

ab 9b         if (b < l or r < a) return {0, LINF};
9c ee         int m = (l+r)/2;
01 ee         return merge(query(a, b, 2*p, l, m), query(a, b, 2*p+1,
m+1, r));
3b cb     }
da 07     pair<int, ll> update(int a, int b, int x, int p=1, int l=0,
int r=n-1) {
f1 6b         prop(p, l, r);
26 9a         if (a <= l and r <= b) {
32 b9             lazy[p] += x;
ac 6b             prop(p, l, r);
42 53             return seg[p];
81 cb         }
ca e9         if (b < l or r < a) return seg[p];
21 ee         int m = (l+r)/2;
d5 08         return seg[p] = merge(update(a, b, x, 2*p, l, m),
14 57             update(a, b, x, 2*p+1, m+1, r));
29 cb     }
04 21 };

ea eb ll seg_vec[MAX];

0c 8b ll area_sq(vector<pair<pair<int, int>, pair<int, int>>> &sq){
1e 28     vector<pair<pair<int, int>, pair<int, int>>> up;
50 60     for (auto it : sq){
b5 61         int x1, y1, x2, y2;
a9 ae         tie(x1, y1) = it.first;
fa 68         tie(x2, y2) = it.second;
ae 80         up.push_back({{x1+1, 1}, {y1, y2}});
ee ae         up.push_back({{x2+1, -1}, {y1, y2}});
6c cb     }
c7 09     sort(up.begin(), up.end());
96 04     memset(seg_vec, 0, sizeof seg_vec);
89 6f     ll H_MAX = MAX;
fb 15     seg::build(H_MAX-1, seg_vec);
0f 7b     auto it = up.begin();
19 04     ll ans = 0;
0f f1     while (it != up.end()){
8d 07         ll L = (*it).first.first;
ba 71         while (it != up.end() && (*it).first.first == L){
f9 12             int x, inc, y1, y2;
45 d3             tie(x, inc) = it->first;
ec d3             tie(y1, y2) = it->second;
8e 5d             seg::update(y1+1, y2, inc);
5c 40             it++;
96 cb         }
2f 85         if (it == up.end()) break;

```

```

d6 d8      ll R = (*it).first.first;

af f5      ll W = R-L;
6f ef      auto jt = seg::query(0, H_MAX-1);
a3 91      ll H = H_MAX - 1;
a5 e8      if (jt.second == 0) H -= jt.first;
4c 8d      ans += W*H;
4d cb      }
43 ba      return ans;
5d cb      }

```

## 2.5 Area Maxima de Histograma

```

// Assume que todas as barras tem largura 1,
// e altura dada no vetor v
//
// O(n)
// e43846

```

```

15 15 ll area(vector<int> v) {
ab b7      ll ret = 0;
70 4c      stack<int> s;
           // valores iniciais pra dar tudo certo
d4 44      v.insert(v.begin(), -1);
dc d5      v.insert(v.end(), -1);
74 1f      s.push(0);

7a 0b      for(int i = 0; i < (int) v.size(); i++) {
e9 78          while (v[s.top()] > v[i]) {
9c 26              ll h = v[s.top()]; s.pop();
fb de              ret = max(ret, h * (i - s.top() - 1));
c3 cb          }
d0 18          s.push(i);
45 cb      }

d0 ed      return ret;
e4 cb      }

```

## 2.6 Binomial modular

```

// Computa C(n, k) mod m em O(m + log(m) log(n))
// = O(rapido)
// ed4344

```

```

97 97 ll divi[MAX];

85 39 ll expo(ll a, ll b, ll m) {

```

```

07 1c      if (!b) return 1;
97 39      ll ans = expo(a*a%m, b/2, m);
7e 75      if (b%2) ans *= a;
ce 2e      return ans%m;
2e cb      }

86 f0 ll inv(ll a, ll b){
ef bc      return 1<a ? b - inv(b%a,a)*b/a : 1;
e0 cb      }

87 15 template<typename T> tuple<T, T, T> ext_gcd(T a, T b) {
3a 3b      if (!a) return {b, 0, 1};
50 55      auto [g, x, y] = ext_gcd(b%a, a);
48 c5      return {g, y - b/a*x, x};
c4 cb      }

22 bf template<typename T = ll> struct crt {
be 62      T a, m;

2b 5f      crt() : a(0), m(1) {}
89 7e      crt(T a_, T m_) : a(a_), m(m_) {}
34 91      crt operator * (crt C) {
41 23          auto [g, x, y] = ext_gcd(m, C.m);
bc dc          if ((a - C.a) % g) a = -1;
e1 4f          if (a == -1 or C.a == -1) return crt(-1, 0);
03 d0          T lcm = m/g*C.m;
29 eb          T ans = a + (x*(C.a-a)/g % (C.m/g))*m;
4e d8          return crt((ans % lcm + lcm) % lcm, lcm);
7b cb      }
32 21 };

45 6f pair<ll, ll> divide_show(ll n, int p, int k, int pak) {
c0 4f      if (n == 0) return {0, 1};
c2 d0      ll blocos = n/pak, falta = n%pak;
51 2c      ll periodo = divi[pak], resto = divi[falta];
c0 61      ll r = expo(periodo, blocos, pak)*resto%pak;

0c 44      auto rec = divide_show(n/p, p, k, pak);
a9 a5      ll y = n/p + rec.first;
18 bb      r = r*rec.second % pak;

18 90      return {y, r};
f4 cb      }

ba 6e ll solve_pak(ll n, ll x, int p, int k, int pak) {
63 d3      divi[0] = 1;
f9 f2      for (int i = 1; i <= pak; i++) {

```

```

b1 90      divi[i] = divi[i-1];
f8 84      if (i%p) divi[i] = divi[i] * i % pak;
f7 cb    }

ce 4a      auto dn = divide_show(n, p, k, pak), dx = divide_show(x, p, k,
    pak),
fc 16      dnx = divide_show(n-x, p, k, pak);
6d 76      ll y = dn.first-dx.first-dnx.first, r =
13 b6      (dn.second*inv(dx.second, pak)%pak)*inv(dnx.second,
    pak)%pak;
71 03      return expo(p, y, pak) * r % pak;
16 cb    }

2b 9d ll solve(ll n, ll x, int mod) {
3d 49      vector<pair<int, int>> f;
eb c3      int mod2 = mod;
77 7b      for (int i = 2; i*i <= mod2; i++) if (mod2%i==0) {
ad af          int c = 0;
2e 75          while (mod2%i==0) mod2 /= i, c++;
26 2a          f.push_back({i, c});
24 cb      }
27 0f      if (mod2 > 1) f.push_back({mod2, 1});
5f e9      crt ans(0, 1);
eb a1      for (int i = 0; i < f.size(); i++) {
dc 70          int pak = 1;
25 7e          for (int j = 0; j < f[i].second; j++) pak *= f[i].first;
e6 30          ans = ans * crt(solve_pak(n, x, f[i].first, f[i].second,
    pak), pak);
5e cb      }
38 5f      return ans.a;
ed cb    }

```

## 2.7 Closest pair of points

```

// O(nlogn)
// f90265

91 91 pair<pt, pt> closest_pair_of_points(vector<pt> v) {
7c 3d      int n = v.size();
e3 fc      sort(v.begin(), v.end());
20 31      for (int i = 1; i < n; i++) if (v[i] == v[i-1]) return
    {v[i-1], v[i]};
77 c2      auto cmp_y = [&](const pt &l, const pt &r) {
54 b5          if (l.y != r.y) return l.y < r.y;
37 92          return l.x < r.x;
dd 21      };
1d 62      set<pt, decltype(cmp_y)> s(cmp_y);

```

```

9b 3d      int l = 0, r = -1;
33 6a      ll d2_min = numeric_limits<ll>::max();
5c 4d      pt pl, pr;
05 bd      const int magic = 5;
70 a5      while (r+1 < n) {
15 7f          auto it = s.insert(v[++r]).first;
bb c9          int cnt = magic/2;
09 77          while (cnt-- and it != s.begin()) it--;
13 a0          cnt = 0;
71 d6          while (cnt++ < magic and it != s.end()) {
8e f1              if (!((*it) == v[r])) {
21 67                  ll d2 = dist2(*it, v[r]);
42 74                  if (d2_min > d2) {
db 22                      d2_min = d2;
c2 84                      pl = *it;
e3 4f                      pr = v[r];
76 cb                  }
5f cb              }
fe 40              it++;
6f cb          }
ed eb          while (l < r and sq(v[l].x-v[r].x) > d2_min)
    s.erase(v[l++]);
f2 cb      }
a6 c7      return {pl, pr};
f9 cb    }

```

## 2.8 Coloracao de Grafo de Intervalo

```

// Colore os intervalos com o numero minimo
// de cores de tal forma que dois intervalos
// que se interceptam tem cores diferentes
// As cores vao de 1 ate n
//
// O(n log(n))
// 83a32d

61 61 vector<int> coloring(vector<pair<int, int>>& v) {
9f 3d      int n = v.size();
f5 c0      vector<pair<int, pair<int, int>>> ev;
45 60      for (int i = 0; i < n; i++) {
39 15          ev.push_back({v[i].first, {1, i}});
61 cd          ev.push_back({v[i].second, {0, i}});
ae cb      }
50 49      sort(ev.begin(), ev.end());
b1 36      vector<int> ans(n), avl(n);
75 26      for (int i = 0; i < n; i++) avl.push_back(n-i);
4c 4b      for (auto i : ev) {

```

```

f0 cb      if (i.second.first == 1) {
65 02          ans[i.second.second] = avl.back();
a4 a0          avl.pop_back();
ac 29      } else avl.push_back(ans[i.second.second]);
62 cb  }
19 ba      return ans;
83 cb  }

```

## 2.9 Conectividade Dinamica

```

// Offline com Divide and Conquer e
// DSU com rollback
// O(n log^2(n))
// 043d93

```

```

8f 8f typedef pair<int, int> T;

51 1c namespace data {
5b 55     int n, ans;
b0 57     int p[MAX], sz[MAX];
a5 ee     stack<int> S;

d3 e5     void build(int n2) {
6a 1e         n = n2;
72 8a         for (int i = 0; i < n; i++) p[i] = i, sz[i] = 1;
cf 0b         ans = n;
8f cb     }
6b 1b     int find(int k) {
7b 00         while (p[k] != k) k = p[k];
a7 83         return k;
1c cb     }
c3 07     void add(T x) {
da 70         int a = x.first, b = x.second;
f5 60         a = find(a), b = find(b);
2a 84         if (a == b) return S.push(-1);
79 e7         ans--;
96 3c         if (sz[a] > sz[b]) swap(a, b);
b0 4c         S.push(a);
f9 58         sz[b] += sz[a];
97 84         p[a] = b;
da cb     }
5e 5e     int query() {
5d ba         return ans;
f9 cb     }
76 5c     void rollback() {
10 46         int u = S.top(); S.pop();
d9 61         if (u == -1) return;

```

```

24 27         sz[p[u]] -= sz[u];
0f 54         p[u] = u;
b3 0d         ans++;
53 cb     }
1a 21 };

e2 35 int ponta[MAX]; // outra ponta do intervalo ou -1 se for query
c7 4f int ans[MAX], n, q;
1a 48 T qu[MAX];

70 47 void solve(int l = 0, int r = q-1) {
56 0b     if (l >= r) {
f8 8c         ans[l] = data::query(); // agora a estrutura ta certa
94 50         return;
6b cb     }
5a 96     int m = (l+r)/2, qnt = 1;
b9 fc     for (int i = m+1; i <= r; i++) if (ponta[i]+1 and ponta[i] < l)
32 37         data::add(qu[i]), qnt++;
fe 22     solve(l, m);
5a 59     while (--qnt) data::rollback();
d4 a2     for (int i = l; i <= m; i++) if (ponta[i]+1 and ponta[i] > r)
40 37         data::add(qu[i]), qnt++;
f9 37     solve(m+1, r);
25 28     while (qnt--) data::rollback();
04 cb }

```

## 2.10 Conectividade Dinamica 2

```

// Offline com link-cut trees
// O(n log(n))
// d38e4e

```

```

1e 1e namespace lct {
c5 3c     struct node {
8b 19         int p, ch[2];
57 a2         int val, sub;
67 aa         bool rev;
d9 f9         node() {}
42 54         node(int v) : p(-1), val(v), sub(v), rev(0) { ch[0] =
ch[1] = -1; }
82 21     };

56 c5     node t[2*MAX]; // MAXN + MAXQ
d9 99     map<pair<int, int>, int> aresta;
13 e4     int sz;

2f 95     void prop(int x) {

```

```

60 aa     if (t[x].rev) {
99 f9         swap(t[x].ch[0], t[x].ch[1]);
64 37         if (t[x].ch[0]+1) t[t[x].ch[0]].rev ^= 1;
fa c3         if (t[x].ch[1]+1) t[t[x].ch[1]].rev ^= 1;
1b cb     }
1f 69     t[x].rev = 0;
e1 cb     }
61 56     void update(int x) {
1f e8         t[x].sub = t[x].val;
37 8c         for (int i = 0; i < 2; i++) if (t[x].ch[i]+1) {
53 62             prop(t[x].ch[i]);
7a 78             t[x].sub = min(t[x].sub, t[t[x].ch[i]].sub);
34 cb         }
3a cb     }
46 97     bool is_root(int x) {
6d 65         return t[x].p == -1 or (t[t[x].p].ch[0] != x and
            t[t[x].p].ch[1] != x);
4b cb     }
91 ed     void rotate(int x) {
59 49         int p = t[x].p, pp = t[p].p;
93 fc         if (!is_root(p)) t[pp].ch[t[pp].ch[1] == p] = x;
f7 25         bool d = t[p].ch[0] == x;
9d 46         t[p].ch[!d] = t[x].ch[d], t[x].ch[d] = p;
2f a7         if (t[p].ch[!d]+1) t[t[p].ch[!d]].p = p;
00 8f         t[x].p = pp, t[p].p = x;
98 44         update(p), update(x);
8f cb     }
9d 23     int splay(int x) {
87 18         while (!is_root(x)) {
60 49             int p = t[x].p, pp = t[p].p;
bc 77             if (!is_root(p)) prop(pp);
21 be             prop(p), prop(x);
b7 0c             if (!is_root(p)) rotate((t[pp].ch[0] == p)^(t[p].ch[0]
            == x) ? x : p);
ce 64             rotate(x);
a3 cb         }
0b aa         return prop(x), x;
25 cb     }
97 f1     int access(int v) {
d8 0e         int last = -1;
db d9         for (int w = v; w+1; update(last = w), splay(v), w =
            t[v].p)
14 02             splay(w), t[w].ch[1] = (last == -1 ? -1 : v);
b3 3d         return last;
b7 cb     }
93 95     void make_tree(int v, int w=INF) { t[v] = node(w); }
f0 82     bool conn(int v, int w) {

```

```

b1 2c         access(v), access(w);
6e b9         return v == w ? true : t[v].p != -1;
4b cb     }
ef 27     void rootify(int v) {
e1 5e         access(v);
73 a0         t[v].rev ^= 1;
1a cb     }
12 a1     int query(int v, int w) {
6b b5         rootify(w), access(v);
45 24         return t[v].sub;
bd cb     }
a3 20     void link_(int v, int w) {
ec 82         rootify(w);
a0 38         t[w].p = v;
8b cb     }
0f 6b     void link(int v, int w, int x) { // v--w com peso x
1b 37         int id = MAX + sz++;
1a 11         aresta[make_pair(v, w)] = id;
c0 ab         make_tree(id, x);
62 c8         link_(v, id), link_(id, w);
95 cb     }
92 e6     void cut_(int v, int w) {
71 b5         rootify(w), access(v);
6d 26         t[v].ch[0] = t[t[v].ch[0]].p = -1;
c3 cb     }
85 03     void cut(int v, int w) {
6b b0         int id = aresta[make_pair(v, w)];
28 a4         cut_(v, id), cut_(id, w);
12 cb     }
0d cb }

8f 89     void dyn_conn() {
88 c5         int n, q; cin >> n >> q;
e8 d6         vector<int> p(2*q, -1); // outra ponta do intervalo
b6 b4         for (int i = 0; i < n; i++) lct::make_tree(i);
05 fb         vector<pair<int, int>> qu(q);
b1 13         map<pair<int, int>, int> m;
f8 ab         for (int i = 0; i < q; i++) {
6b 3c             char c; cin >> c;
0b ef             if (c == '?') continue;
87 60             int a, b; cin >> a >> b; a--, b--;
be d1             if (a > b) swap(a, b);
33 8a             qu[i] = {a, b};
a5 8d             if (c == '+') {
36 94                 p[i] = i+q, p[i+q] = i;
94 90                 m[make_pair(a, b)] = i;
a7 9d             } else {

```

```

50 41         int j = m[make_pair(a, b)];
70 ac         p[i] = j, p[j] = i;
e1 cb     }
4d cb }
19 44 int ans = n;
4f ab for (int i = 0; i < q; i++) {
4b 87     if (p[i] == -1) {
0a 88         cout << ans << endl; // numero de comp conexos
a4 5e         continue;
aa cb     }
90 69     int a = qu[i].first, b = qu[i].second;
af c4     if (p[i] > i) { // +
d9 ac         if (lct::conn(a, b)) {
77 18             int mi = lct::query(a, b);
6a 99             if (p[i] < mi) {
c9 dd                 p[p[i]] = p[i];
df 5e                 continue;
23 cb             }
89 6f             lct::cut(qu[p[mi]].first, qu[p[mi]].second), ans++;
7e 6e             p[mi] = mi;
3a cb         }
77 d1         lct::link(a, b, p[i]), ans--;
51 cb     } else if (p[i] != i) lct::cut(a, b), ans++; // -
c1 cb }
d3 cb }

```

## 2.11 Conj. Indep. Maximo com Peso em Grafo de Intervalo

```

// Retorna os indices ordenados dos intervalos selecionados
// Se tiver empate, retorna o que minimiza o comprimento total
//
// 0(n log(n))
// c4dbe2

```

```

31 31 vector<int> ind_set(vector<tuple<int, int, int>>& v) {
fb b2     vector<tuple<int, int, int>> w;
9f f1     for (int i = 0; i < v.size(); i++) {
70 e8         w.push_back(tuple(get<0>(v[i]), 0, i));
ef 6f         w.push_back(tuple(get<1>(v[i]), 1, i));
fc cb     }
3d d1     sort(w.begin(), w.end());

5e 84     vector<int> nxt(v.size());
ef c2     vector<pair<ll, int>> dp(v.size());
79 0e     int last = -1;
54 72     for (auto [fim, t, i] : w) {
d1 25         if (t == 0) {

```

```

53 4c         nxt[i] = last;
29 5e         continue;
89 cb     }
bf 78     dp[i] = {0, 0};
e5 cb     if (last != -1) dp[i] = max(dp[i], dp[last]);
38 91     pair<ll, int> pega = {get<2>(v[i]), -(get<1>(v[i]) -
get<0>(v[i]) + 1)};
71 5d     if (nxt[i] != -1) pega.first += dp[nxt[i]].first,
pega.second += dp[nxt[i]].second;
0e b0     if (pega > dp[i]) dp[i] = pega;
e3 7c     else nxt[i] = last;
12 38     last = i;
98 cb }
51 97     pair<ll, int> ans = {0, 0};
00 91     int idx = -1;
91 ce     for (int i = 0; i < v.size(); i++) if (dp[i] > ans) ans =
dp[i], idx = i;
1e 4b     vector<int> ret;
3e fd     while (idx != -1) {
05 d6         if (get<2>(v[idx]) > 0 and
6e a0             (nxt[idx] == -1 or get<1>(v[nxt[idx]]) <
get<0>(v[idx]))) ret.push_back(idx);
d1 e4         idx = nxt[idx];
58 cb     }
e2 0e     sort(ret.begin(), ret.end());
d5 ed     return ret;
c4 cb }

```

## 2.12 Distancia maxima entre dois pontos

```

// max_dist2(v) - O(n log(n))
// max_dist_manhattan - O(n)

```

```

// Quadrado da Distancia Euclidiana (precisa copiar convex_hull, ccw e
pt)
// bdace4
85 85 ll max_dist2(vector<pt> v) {
bc 22     v = convex_hull(v);
78 a1     if (v.size() <= 2) return dist2(v[0], v[1%v.size()]);
d6 04     ll ans = 0;
55 32     int n = v.size(), j = 0;
78 60     for (int i = 0; i < n; i++) {
d8 05         while (!ccw(v[(i+1)%n]-v[i], pt(0, 0), v[(j+1)%n]-v[j])) j
= (j+1)%n;
21 e7         ans = max({ans, dist2(v[i], v[j]), dist2(v[(i+1)%n],
v[j])});
40 cb     }

```

```

53 ba    return ans;
bd cb }

// Distancia de Manhattan
// 4e96f0
82 c5 template<typename T> T max_dist_manhattan(vector<pair<T, T>> v) {
e7 8e    T min_sum, max_sum, min_dif, max_dif;
91 4f    min_sum = max_sum = v[0].first + v[0].second;
a7 27    min_dif = max_dif = v[0].first - v[0].second;
71 c2    for (auto [x, y] : v) {
44 1c        min_sum = min(min_sum, x+y);
77 68        max_sum = max(max_sum, x+y);
4b 78        min_dif = min(min_dif, x-y);
d9 af        max_dif = max(max_dif, x-y);
66 cb    }
56 9f    return max(max_sum - min_sum, max_dif - min_dif);
ce cb }

```

## 2.13 Distinct Range Query

```

// build - O(n (log n + log(sigma)))
// query - O(log(sigma))
// 5c7aa1

78 78 namespace perseg { };

6c 53 int qt[MAX];

46 ed void build(vector<int>& v) {
c8 3d    int n = v.size();
c2 16    perseg::build(n);
30 66    map<int, int> last;
a4 05    int at = 0;
9f 60    for (int i = 0; i < n; i++) {
2a 81        if (last.count(v[i])) {
56 a5            perseg::update(last[v[i]], -1);
b3 69            at++;
43 cb        }
81 4f        perseg::update(i, 1);
75 46        qt[i] = ++at;
fe ef        last[v[i]] = i;
20 cb    }
b1 cb }

e0 9e int query(int l, int r) {
2e 08    return perseg::query(l, r, qt[r]);
5c cb }

```

## 2.14 Distinct Range Query com Update

```

// build - O(n log(n))
// query - O(log^2(n))
// update - O(log^2(n))
// 2306f3

77 77 #include <ext/pb_ds/assoc_container.hpp>
07 30 #include <ext/pb_ds/tree_policy.hpp>
99 0d using namespace __gnu_pbds;
e5 4f template <class T>
bd de    using ord_set = tree<T, null_type, less<T>, rb_tree_tag,
90 3a    tree_order_statistics_node_update>;

b5 04 int v[MAX], n, nxt[MAX], prv[MAX];
c9 f6 map<int, set<int> > oco;

0a e0 namespace bit {
be 68    ord_set<pair<int, int>> bit[MAX];

a2 0a    void build() {
44 3e        for (int i = 1; i <= n; i++) bit[i].insert({nxt[i-1],
i-1});
2a 78        for (int i = 1; i <= n; i++) {
69 ed            int j = i + (i&-i);
4f d0            if (j <= n) for (auto x : bit[i]) bit[j].insert(x);
a5 cb        }
be cb    }
b8 d3    int pref(int p, int x) {
6c 7c        int ret = 0;
8e bb        for (; p; p -= p&-p) ret += bit[p].order_of_key({x, -INF});
0d ed        return ret;
fc cb    }
9c d5    int query(int l, int r, int x) {
bc e5        return pref(r+1, x) - pref(l, x);
c9 cb    }
08 ff    void update(int p, int x) {
07 f1        int p2 = p;
cb 5e        for (p++; p <= n; p += p&-p) {
8e ca            bit[p].erase({nxt[p2], p2});
d5 f6            bit[p].insert({x, p2});
96 cb        }
06 cb    }
23 cb }

19 0a void build() {
0b 38    for (int i = 0; i < n; i++) nxt[i] = INF;

```



```

db 7b   for (int i = 0; i < n; i++) prv[i] = -INF;
04 d0   vector<pair<int, int>> t;
c0 34   for (int i = 0; i < n; i++) t.push_back({v[i], i});
bb 3f   sort(t.begin(), t.end());
83 60   for (int i = 0; i < n; i++) {
f1 b4       if (i and t[i].first == t[i-1].first)
b4 56         prv[t[i].second] = t[i-1].second;
97 a8       if (i+1 < n and t[i].first == t[i+1].first)
c1 12         nxt[t[i].second] = t[i+1].second;
f1 cb   }

78 a2   for (int i = 0; i < n; i++) ocor[v[i]].insert(i);

79 1d   bit::build();
13 cb }

31 aa void muda(int p, int x) {
f9 f9     bit::update(p, x);
1d c3     nxt[p] = x;
89 cb }

01 4e int query(int a, int b) {
ca a0     return b-a+1 - bit::query(a, b, b+1);
1e cb }

95 ff void update(int p, int x) { // mudar valor na pos. p para x
9a c0     if (prv[p] > -INF) muda(prv[p], nxt[p]);
65 4a     if (nxt[p] < INF) prv[nxt[p]] = prv[p];

ef 5b     ocor[v[p]].erase(p);
69 4b     if (!ocor[x].size()) {
67 19         muda(p, INF);
93 8d         prv[p] = -INF;
06 a6     } else if (*ocor[x].rbegin() < p) {
58 5b         int i = *ocor[x].rbegin();
58 f6         prv[p] = i;
f3 19         muda(p, INF);
0a 5f         muda(i, p);
17 9d     } else {
ec d4         int i = *ocor[x].lower_bound(p);
57 33         if (prv[i] > -INF) {
ca f1             muda(prv[i], p);
96 8f             prv[p] = prv[i];
8a 94         } else prv[p] = -INF;
51 52         prv[i] = p;
ea 59         muda(p, i);
79 cb     }

```

```

65 c9     v[p] = x; ocor[x].insert(p);
23 cb }

```

## 2.15 Dominator Points

```

// Se um ponto A tem ambas as coordenadas >= B, dizemos
// que A domina B
// is_dominated(p) fala se existe algum ponto no conjunto
// que domina p
// insert(p) insere p no conjunto
// (se p for dominado por alguem, nao vai inserir)
// o multiset 'quina' guarda informacao sobre os pontos
// nao dominados por um elemento do conjunto que nao dominam
// outro ponto nao dominado por um elemento do conjunto
// No caso, armazena os valores de x+y esses pontos
//
// Complexidades:
// is_dominated - O(log(n))
// insert - O(log(n)) amortizado
// query - O(1)
// 09ffdc

```

```

e2 e2 struct dominator_points {
3e ba     set<pair<int, int>> se;
40 4d     multiset<int> quina;

98 a8     bool is_dominated(pair<int, int> p) {
9c 80         auto it = se.lower_bound(p);
da 63         if (it == se.end()) return 0;
96 ab         return it->second >= p.second;
8a cb     }
ac 99     void mid(pair<int, int> a, pair<int, int> b, bool rem) {
61 29         pair<int, int> m = {a.first+1, b.second+1};
71 b1         int val = m.first + m.second;
c5 63         if (!rem) quina.insert(val);
ef 73         else quina.erase(quina.find(val));
8d cb     }
86 7c     bool insert(pair<int, int> p) {
8b fb         if (is_dominated(p)) return 0;
ae 80         auto it = se.lower_bound(p);
4b ca         if (it != se.begin() and it != se.end())
1a d4             mid(*prev(it), *it, 1);
c4 1f         while (it != se.begin()) {
60 04             it--;
ab 23             if (it->second > p.second) break;
0e b8             if (it != se.begin()) mid(*prev(it), *it, 1);
93 31             it = se.erase(it);

```

```

46 cb      }
a0 43      it = se.insert(p).first;
58 69      if (it != se.begin()) mid(*prev(it), *it, 0);
ba 96      if (next(it) != se.end()) mid(*it, *next(it), 0);
52 6a      return 1;
cd cb    }
34 5e    int query() {
64 95      if (!quina.size()) return INF;
19 ad      return *quina.begin();
3d cb    }
09 21 };

```

## 2.16 DP de Dominacao 3D

```

// Computa para todo ponto i,
// dp[i] = 1 + max_{j dominado por i} dp[j]
// em que ser dominado eh ter as 3 coordenadas menores
// Da pra adaptar facil para outras dps
//
// O(n log^2 n), O(n) de memoria
// 7c8896

c5 c5 void lis2d(vector<vector<tuple<int, int, int>>>& v, vector<int>&
dp, int l, int r) {
24 89     if (l == r) {
7d 56         for (int i = 0; i < v[l].size(); i++) {
c9 8b             int ii = get<2>(v[l][i]);
c6 1c             dp[ii] = max(dp[ii], 1);
50 cb         }
a3 50         return;
7b cb     }
b5 ee     int m = (l+r)/2;
2a 62     lis2d(v, dp, l, m);

ad 32     vector<tuple<int, int, int>> vv[2];
6c d4     vector<int> Z;
21 87     for (int i = l; i <= r; i++) for (auto it : v[i]) {
e8 2e         vv[i > m].push_back(it);
b4 04         Z.push_back(get<1>(it));
e8 cb     }
43 e9     sort(vv[0].begin(), vv[0].end());
27 9b     sort(vv[1].begin(), vv[1].end());
e8 0d     sort(Z.begin(), Z.end());
55 57     auto get_z = [&](int z) { return lower_bound(Z.begin(),
Z.end(), z) - Z.begin(); };
79 c5     vector<int> bit(Z.size());

```

```

f2 18     int i = 0;
f2 e9     for (auto [y, z, id] : vv[1]) {
6a 6b         while (i < vv[0].size() and get<0>(vv[0][i]) < y) {
77 39             auto [y2, z2, id2] = vv[0][i++];
ec ea             for (int p = get_z(z2)+1; p <= Z.size(); p += p&-p)
47 30                 bit[p-1] = max(bit[p-1], dp[id2]);
7e cb         }
6d d3         int q = 0;
f2 fd         for (int p = get_z(z); p; p -= p&-p) q = max(q, bit[p-1]);
ba 61         dp[id] = max(dp[id], q + 1);
fa cb     }
59 c2     lis2d(v, dp, m+1, r);
4d cb }

69 4d vector<int> solve(vector<tuple<int, int, int>> v) {
f1 3d     int n = v.size();
97 cd     vector<tuple<int, int, int, int>> vv;
f9 60     for (int i = 0; i < n; i++) {
74 9b         auto [x, y, z] = v[i];
c7 5b         vv.emplace_back(x, y, z, i);
40 cb     }
0e bd     sort(vv.begin(), vv.end());

a2 e1     vector<vector<tuple<int, int, int>>> V;
d6 60     for (int i = 0; i < n; i++) {
b5 a5         int j = i;
bd 80         V.emplace_back();
19 c0         while (j < n and get<0>(vv[j]) == get<0>(vv[i])) {
b9 ba             auto [x, y, z, id] = vv[j++];
f8 cb             V.back().emplace_back(y, z, id);
7b cb         }
52 45         i = j-1;
15 cb     }
aa 38     vector<int> dp(n);
c0 83     lis2d(V, dp, 0, V.size()-1);
d9 89     return dp;
7c cb }

```

## 2.17 Gray Code

```

// Gera uma permutacao de 0 a 2^n-1, de forma que
// duas posicoes adjacentes diferem em exatamente 1 bit
//
// O(2^n)
// 840df4

df df vector<int> gray_code(int n) {

```

```

86 73    vector<int> ret(1<<n);
e3 f2    for (int i = 0; i < (1<<n); i++) ret[i] = i^(i>>1);
68 ed    return ret;
84 cb }

```

## 2.18 Half-plane intersection

```

// Cada half-plane eh identificado por uma reta e a regioao ccw a ela
//
// O(n log n)
// f56e1c

```

```

f4 f4 vector<pt> hp_intersection(vector<line> &v) {
5c 9b    deque<pt> dq = {{INF, INF}, {-INF, INF}, {-INF, -INF}, {INF,
    -INF}};

```

```

5c d4 #warning considerar trocar por compare_angle
4a de    sort(v.begin(), v.end(), [&](line r, line s) { return
    angle(r.q-r.p) < angle(s.q-s.p); });

```

```

66 5e    for(int i = 0; i < v.size() and dq.size() > 1; i++) {
c8 c6        pt p1 = dq.front(), p2 = dq.back();
7c 6c        while (dq.size() and !ccw(v[i].p, v[i].q, dq.back()))
1b 47            p1 = dq.back(), dq.pop_back();
a9 0a        while (dq.size() and !ccw(v[i].p, v[i].q, dq.front()))
17 7c            p2 = dq.front(), dq.pop_front();

```

```

b7 4d        if (!dq.size()) break;
13 60        if (p1 == dq.front() and p2 == dq.back()) continue;
e4 c9        dq.push_back(inter(v[i], line(dq.back(), p1)));
e2 65        dq.push_front(inter(v[i], line(dq.front(), p2)));

```

```

e1 fd        if (dq.size() > 1 and dq.back() == dq.front())
    dq.pop_back();
db cb    }
b4 b2    return vector<pt>(dq.begin(), dq.end());
f5 cb }

```

## 2.19 Heap Sort

```

// O(n log n)
// 385e91

```

```

f1 f1 void down(vector<int>& v, int n, int i) {
a6 e1    while ((i = 2*i+1) < n) {
8a 58        if (i+1 < n and v[i] < v[i+1]) i++;
32 b2        if (v[i] < v[(i-1)/2]) break;

```

```

54 32        swap(v[i], v[(i-1)/2]);
6d cb    }
72 cb }
9a eb void heap_sort(vector<int>& v) {
92 3d    int n = v.size();
e1 61    for (int i = n/2-1; i >= 0; i--) down(v, n, i);
7d 91    for (int i = n-1; i > 0; i--)
e1 37        swap(v[0], v[i]), down(v, i, 0);
38 cb }

```

## 2.20 Inversion Count

```

// Computa o numero de inversoes para transformar
// l em r (se nao tem como, retorna -1)
//
// O(n log(n))
// eef01f

```

```

37 37 template<typename T> ll inv_count(vector<T> l, vector<T> r = {})
    {
71 bb    if (!r.size()) {
ce 79        r = l;
9c 1b        sort(r.begin(), r.end());
ed cb    }
d4 87    int n = l.size();
1d 8c    vector<int> v(n), bit(n);
1a 4e    vector<pair<T, int>> w;
a6 61    for (int i = 0; i < n; i++) w.push_back({r[i], i+1});
e9 d1    sort(w.begin(), w.end());
d3 60    for (int i = 0; i < n; i++) {
ef bf        auto it = lower_bound(w.begin(), w.end(), make_pair(l[i],
    0));
85 1b        if (it == w.end() or it->first != l[i]) return -1; // nao
    da
65 96        v[i] = it->second;
a8 6c        it->second = -1;
4c cb    }

```

```

99 04    ll ans = 0;
7d 45    for (int i = n-1; i >= 0; i--) {
8e 2d        for (int j = v[i]-1; j; j -= j&-j) ans += bit[j];
73 3a        for (int j = v[i]; j < n; j += j&-j) bit[j]++;
c2 cb    }
c7 ba    return ans;
ee cb }

```

## 2.21 LIS - Longest Increasing Subsequence

```
// Calcula e retorna uma LIS
//
// O(n.log(n))
// 4749e8

12 12 template<typename T> vector<T> lis(vector<T>& v) {
3b 1f     int n = v.size(), m = -1;
22 f0     vector<T> d(n+1, INF);
36 ae     vector<int> l(n);
d6 00     d[0] = -INF;

75 60     for (int i = 0; i < n; i++) {
// Para non-decreasing use upper_bound()
7f 4f         int t = lower_bound(d.begin(), d.end(), v[i]) - d.begin();
13 3a         d[t] = v[i], l[i] = t, m = max(m, t);
11 cb     }

81 4f     int p = n;
1a 5a     vector<T> ret;
37 cd     while (p-- > 0) if (l[p] == m) {
c4 88         ret.push_back(v[p]);
20 76         m--;
6e cb     }
4a 96     reverse(ret.begin(), ret.end());

06 ed     return ret;
47 cb }
```

## 2.22 LIS2 - Longest Increasing Subsequence

```
// Calcula o tamanho da LIS
//
// O(n log(n))
// 402def

84 84 template<typename T> int lis(vector<T> &v){
ce 2d     vector<T> ans;
9b 5e     for (T t : v){
// Para non-decreasing use upper_bound()
84 fe         auto it = lower_bound(ans.begin(), ans.end(), t);
73 d7         if (it == ans.end()) ans.push_back(t);
e3 b9         else *it = t;
40 cb     }
14 1e     return ans.size();
40 cb }
```

## 2.23 Minimum Enclosing Circle

```
// O(n) com alta probabilidade
// b0a6ba

22 22 const double EPS = 1e-12;
d7 87 mt19937 rng((int)
    chrono::steady_clock::now().time_since_epoch().count());

ab b2 struct pt {
dc 66     double x, y;
34 be     pt(double x_ = 0, double y_ = 0) : x(x_), y(y_) {}
9c 7a     pt operator + (const pt& p) const { return pt(x+p.x, y+p.y); }
f4 b2     pt operator - (const pt& p) const { return pt(x-p.x, y-p.y); }
b9 25     pt operator * (double c) const { return pt(x*c, y*c); }
cf 70     pt operator / (double c) const { return pt(x/c, y/c); }
d3 21 };

ee 2f double dot(pt p, pt q) { return p.x*q.x+p.y*q.y; }
72 dd double cross(pt p, pt q) { return p.x*q.y-p.y*q.x; }
1a e7 double dist(pt p, pt q) { return sqrt(dot(p-q, p-q)); }

dd 3f pt center(pt p, pt q, pt r) {
cf 5d     pt a = p-r, b = q-r;
89 e8     pt c = pt(dot(a, p+r)/2, dot(b, q+r)/2);
65 e0     return pt(cross(c, pt(a.y, b.y)), cross(pt(a.x, b.x), c)) /
        cross(a, b);
be cb }

0a aa struct circle {
74 f4     pt cen;
57 c1     double r;
4c 89     circle(pt cen_, double r_) : cen(cen_), r(r_) {}
44 83     circle(pt a, pt b, pt c) {
08 13         cen = center(a, b, c);
8e 1f         r = dist(cen, a);
b2 cb     }
f9 cd     bool inside(pt p) { return dist(p, cen) < r+EPS; }
a5 21 };

f8 80 circle minCirc(vector<pt> v) {
6c f2     shuffle(v.begin(), v.end(), rng);
4a ae     circle ret = circle(pt(0, 0), 0);
ef 61     for (int i = 0; i < v.size(); i++) if (!ret.inside(v[i])) {
9a 16         ret = circle(v[i], 0);
34 f1         for (int j = 0; j < i; j++) if (!ret.inside(v[j])) {
54 88             ret = circle((v[i]+v[j])/2, dist(v[i], v[j])/2);
        }
```

```
fb b8         for (int k = 0; k < j; k++) if (!ret.inside(v[k]))
f0 43         ret = circle(v[i], v[j], v[k]);
12 cb     }
b4 cb     }
9f ed     return ret;
b0 cb }
```

## 2.24 Minkowski Sum

```
// Computa A+B = {a+b : a \in A, b \in B}, em que
// A e B sao poligonos convexos
// A+B eh um poligono convexo com no max |A|+|B| pontos
//
// O(|A|+|B|)

// d7cca8
53 53 vector<pt> minkowski(vector<pt> p, vector<pt> q) {
4c 05     auto fix = [](vector<pt>& P) {
29 51         rotate(P.begin(), min_element(P.begin(), P.end()),
        P.end());
12 01         P.push_back(P[0]), P.push_back(P[1]);
5d 21     };
b3 88     fix(p), fix(q);
7a 8a     vector<pt> ret;
c6 69     int i = 0, j = 0;
f7 2e     while (i < p.size()-2 or j < q.size()-2) {
f4 89         ret.push_back(p[i] + q[j]);
9f 73         auto c = ((p[i+1] - p[i]) ^ (q[j+1] - q[j]));
30 eb         if (c >= 0) i = min<int>(i+1, p.size()-2);
ba 81         if (c <= 0) j = min<int>(j+1, q.size()-2);
9b cb     }
a7 ed     return ret;
d7 cb }

// 2f5dd2
3a c3 ld dist_convex(vector<pt> p, vector<pt> q) {
e0 dc     for (pt& i : p) i = i * -1;
66 44     auto s = minkowski(p, q);
e0 95     if (inpol(s, pt(0, 0))) return 0;
85 6a     return 1;
65 92     ld ans = DINf;
1c 07     for (int i = 0; i < s.size(); i++) ans = min(ans,
4c f0         disttoseg(pt(0, 0), line(s[(i+1)%s.size()], s[i])));
ed ba     return ans;
12 cb }
```

## 2.25 MO - DSU

```
// Dado uma lista de arestas de um grafo, responde
// para cada query(l, r), quantos componentes conexos
// o grafo tem se soh considerar as arestas l, l+1, ..., r
// Da pra adaptar pra usar MO com qualquer estrutura rollbackavel
//
// O(m sqrt(q) log(n))
// 704722

8d 8d struct dsu {
a7 55     int n, ans;
72 2e     vector<int> p, sz;
c1 ee     stack<int> S;

53 4b     dsu(int n_) : n(n_), ans(n), p(n), sz(n) {
25 8a         for (int i = 0; i < n; i++) p[i] = i, sz[i] = 1;
78 cb     }
d1 1b     int find(int k) {
e3 00         while (p[k] != k) k = p[k];
b8 83         return k;
29 cb     }
23 55     void add(pair<int, int> x) {
fb 70         int a = x.first, b = x.second;
df 60         a = find(a), b = find(b);
b1 84         if (a == b) return S.push(-1);
e9 e7         ans--;
72 3c         if (sz[a] > sz[b]) swap(a, b);
68 4c         S.push(a);
fe 58         sz[b] += sz[a];
6f 84         p[a] = b;
2e cb     }
d9 35     int query() { return ans; }
ee 5c     void rollback() {
7b 46         int u = S.top(); S.pop();
14 61         if (u == -1) return;
79 27         sz[p[u]] -= sz[u];
ca 54         p[u] = u;
fe 0d         ans++;
a2 cb     }
9c 21 };

ce 1a int n;
5e e9 vector<pair<int, int>> ar;

// 9d242b
3f 61 vector<int> MO(vector<pair<int, int>> &q) {
```

```

14 54     int SQ = sqrt(q.size()) + 1;
56 c2     int m = q.size();
ee 3f     vector<int> ord(m);
8f be     iota(ord.begin(), ord.end(), 0);
ce d0     sort(ord.begin(), ord.end(), [&](int l, int r) {
9b 9c         if (q[l].first / SQ != q[r].first / SQ) return
            q[l].first < q[r].first;
0a a6         return q[l].second < q[r].second;
cf c0     });
9b 43     vector<int> ret(m);

7a 3b     dsu small(n);
3b dd     for (int i = 0; i < m; i++) {
90 5e         auto [l, r] = q[ord[i]];
54 ac         if (l / SQ == r / SQ) {
70 00             for (int k = l; k <= r; k++) small.add(ar[k]);
a8 b9             ret[ord[i]] = small.query();
55 64             for (int k = l; k <= r; k++) small.rollback();
cc cb         }
a4 cb     }

9f dd     for (int i = 0; i < m; i++) {
2d 17         dsu D(n);
bd ae         int fim = q[ord[i]].first/SQ*SQ + SQ - 1;
20 e2         int last_r = fim;
ff eb         int j = i-1;
7a 00         while (j+1 < m and q[ord[j+1]].first / SQ ==
            q[ord[i]].first / SQ) {
d9 a0             auto [l, r] = q[ord[++j]];

7e f5             if (l / SQ == r / SQ) continue;

56 59             while (last_r < r) D.add(ar[++last_r]);
ce 2c             for (int k = l; k <= fim; k++) D.add(ar[k]);

8a 9b             ret[ord[j]] = D.query();

ca 57             for (int k = l; k <= fim; k++) D.rollback();
9e cb         }
05 bd         i = j;
56 cb     }
ea ed     return ret;
70 cb }

```

## 2.26 Mo - numero de distintos em range

// Para ter o bound abaixo, escolher

```

// SQ = n / sqrt(q)
//
// 0(n * sqrt(q))
// e94f60

0d 0d     const int MAX = 1e5+10;
4b 6f     const int SQ = sqrt(MAX);
2a b6     int v[MAX];

81 b6     int ans, freq[MAX];

a3 9d     inline void insert(int p) {
b5 ae         int o = v[p];
30 59         freq[o]++;
18 99         ans += (freq[o] == 1);
9f cb     }

a5 a2     inline void erase(int p) {
10 ae         int o = v[p];
da 7e         ans -= (freq[o] == 1);
6f ba         freq[o]--;
0a cb     }

c1 e5     inline ll hilbert(int x, int y) {
85 71         static int N = 1 << (__builtin_clz(0) - __builtin_clz(MAX));
f7 10         int rx, ry, s;
2e b7         ll d = 0;
f2 43         for (s = N/2; s > 0; s /= 2) {
b3 c9             rx = (x & s) > 0, ry = (y & s) > 0;
3f e3             d += s * ll(s) * ((3 * rx) ^ ry);
48 d2             if (ry == 0) {
86 5a                 if (rx == 1) x = N-1 - x, y = N-1 - y;
fe 9d                 swap(x, y);
aa cb             }
2d cb         }
32 be         return d;
87 cb     }

86 ba     #define HILBERT true
0b 61     vector<int> MO(vector<pair<int, int>> &q) {
8e c3         ans = 0;
0f c2         int m = q.size();
3a 3f         vector<int> ord(m);
0a be         iota(ord.begin(), ord.end(), 0);
0d 6a     #if HILBERT
71 8c         vector<ll> h(m);
86 74         for (int i = 0; i < m; i++) h[i] = hilbert(q[i].first,

```

```

    q[i].second);
02 07    sort(ord.begin(), ord.end(), [&](int l, int r) { return h[l] <
    h[r]; });
ef 8c #else
06 d0    sort(ord.begin(), ord.end(), [&](int l, int r) {
79 9c        if (q[l].first / SQ != q[r].first / SQ) return q[l].first
    < q[r].first;
65 0d        if ((q[l].first / SQ) % 2) return q[l].second >
    q[r].second;
6f a6        return q[l].second < q[r].second;
7b c0    });
d4 f2 #endif
59 43    vector<int> ret(m);
0d 3d    int l = 0, r = -1;

59 8b    for (int i : ord) {
a2 6c        int ql, qr;
a6 4f        tie(ql, qr) = q[i];
cf 02        while (r < qr) insert(++r);
d5 23        while (l > ql) insert(--l);
d5 75        while (l < ql) erase(l++);
f3 fe        while (r > qr) erase(r--);
b9 38        ret[i] = ans;
0f cb    }
57 ed    return ret;
e9 cb }

```

## 2.27 Palindromic Factorization

```

// Precisa da eertree
// Computa o numero de formas de particionar cada
// prefixo da string em strings palindromicas
//
// O(n log n), considerando alfabeto O(1)
// 9e6e22

07 07 struct eertree { ... };

3b 0e ll factorization(string s) {
81 b1    int n = s.size(), sz = 2;
1c 58    eertree PT(n);
70 14    vector<int> diff(n+2), slink(n+2), sans(n+2), dp(n+1);
c0 0e    dp[0] = 1;
1e 78    for (int i = 1; i <= n; i++) {
66 c5        PT.add(s[i-1]);
52 a7        if (PT.size()+2 > sz) {
d3 6c            diff[sz] = PT.len[sz] - PT.len[PT.link[sz]];

```

```

19 24        if (diff[sz] == diff[PT.link[sz]])
8b d6            slink[sz] = slink[PT.link[sz]];
40 f5        else slink[sz] = PT.link[sz];
52 eb        sz++;
a4 cb    }
74 91    for (int v = PT.last; PT.len[v] > 0; v = slink[v]) {
bb 29        sans[v] = dp[i - (PT.len[slink[v]] + diff[v])];
d8 85        if (diff[v] == diff[PT.link[v]])
86 f2            sans[v] = (sans[v] + sans[PT.link[v]]) % MOD;
aa 07        dp[i] = (dp[i] + sans[v]) % MOD;
14 cb    }
df cb    }
9f 5f    return dp[n];
9e cb }

```

## 2.28 Parsing de Expressao

```

// Operacoes associativas a esquerda por default
// Para mudar isso, colocar em r_assoc
// Operacoes com maior prioridade sao feitas primeiro
//
// 9ad15a

cc cc bool blank(char c) {
cc f3    return c == ' ';
ec cb }

3f 8e bool is_unary(char c) {
b5 f9    return c == '+' or c == '-';
fd cb }

41 76 bool is_op(char c) {
69 01    if (is_unary(c)) return true;
1e 31    return c == '*' or c == '/' or c == '+' or c == '-';
3d cb }

bb fa bool r_assoc(char op) {
        // operator unario - deve ser assoc. a direita
44 cf    return op < 0;
9e cb }

63 79 int priority(char op) {
        // operator unario - deve ter precedencia maior
42 10    if (op < 0) return INF;

e0 72    if (op == '*' or op == '/') return 2;
98 43    if (op == '+' or op == '-') return 1;

```

```

b8 da    return -1;
99 cb }

b5 c1 void process_op(stack<int>& st, stack<int>& op) {
b9 88     char o = op.top(); op.pop();
6f 91     if (o < 0) {
dd 4e         o *= -1;
14 1e         int l = st.top(); st.pop();
ba 0f         if (o == '+') st.push(l);
cd 7e         if (o == '-') st.push(-l);
e4 9d     } else {
51 14         int r = st.top(); st.pop();
5d 1e         int l = st.top(); st.pop();
99 1e         if (o == '*') st.push(l * r);
f0 f5         if (o == '/') st.push(l / r);
39 60         if (o == '+') st.push(l + r);
5d c4         if (o == '-') st.push(l - r);
c9 cb     }
07 cb }

65 43 int eval(string& s) {
44 21     stack<int> st, op;
4b d0     bool un = true;
f3 1c     for (int i = 0; i < s.size(); i++) {
0d 68         if (blank(s[i])) continue;

be 13         if (s[i] == '(') {
bf 36             op.push('(');
af 99             un = true;
e8 13         } else if (s[i] == ')') {
95 70             while (op.top() != '(') process_op(st, op);
cb 75             op.pop();
44 ce             un = false;
a1 14         } else if (is_op(s[i])) {
3d 4d             char o = s[i];
29 37             if (un and is_unary(o)) o *= -1;
2b ae             while (op.size() and (
de cd                 (!r_assoc(o) and priority(op.top()) >=
                    priority(o)) or
4e c4                 (r_assoc(o) and priority(op.top()) >
                    priority(o))))
e5 c4                 process_op(st, op);
d9 c0                 op.push(o);
2b 99                 un = true;
3f 9d         } else {
a4 da             int val = 0;
75 c2             while (i < s.size() and isalnum(s[i]))

```

```

f7 8a                 val = val * 10 + s[i++] - '0';
68 16                 i--;
30 25                 st.push(val);
0d ce                 un = false;
27 cb             }
34 cb         }

6e 7f     while (op.size()) process_op(st, op);
e7 12     return st.top();
9a cb }

```

## 2.29 RMQ com Divide and Conquer

```

// Responde todas as queries em
// O(n log(n))
// 5a6ebd

f7 f7 typedef pair<pair<int, int>, int> iii;
ea 7c #define f first
ab 0a #define s second

69 87 int n, q, v[MAX];
b3 e3 iii qu[MAX];
56 ae int ans[MAX], pref[MAX], sulf[MAX];

09 0e void solve(int l=0, int r=n-1, int ql=0, int qr=q-1) {
94 8a     if (l > r or ql > qr) return;
4c ee     int m = (l+r)/2;
da 1b     int qL = partition(qu+ql, qu+qr+1, [=](iii x){return x.f.s <
m;}) - qu;
1f eb     int qR = partition(qu+qL, qu+qr+1, [=](iii x){return x.f.f
<=m;}) - qu;

d2 3c     pref[m] = sulf[m] = v[m];
cd 9f     for (int i = m-1; i >= l; i--) pref[i] = min(v[i], pref[i+1]);
04 ea     for (int i = m+1; i <= r; i++) sulf[i] = min(v[i], sulf[i-1]);

ea b2     for (int i = qL; i < qR; i++)
ea f3         ans[qu[i].s] = min(pref[qu[i].f.f], sulf[qu[i].f.s]);

f5 36     solve(l, m-1, ql, qL-1), solve(m+1, r, qR, qr);
5a cb }

```

## 2.30 Segment Intersection

```

// Verifica, dado n segmentos, se existe algum par de segmentos
// que se intersecta

```



```
//
// 0(n log n)
// 3957d8

6e 6e bool operator < (const line& a, const line& b) { // comparador
    pro sweepline
26 19 if (a.p == b.p) return ccw(a.p, a.q, b.q);
3b 23 if (!eq(a.p.x, a.q.x) and (eq(b.p.x, b.q.x) or a.p.x+eps <
    b.p.x))
b8 78 return ccw(a.p, a.q, b.p);
b8 dc return ccw(a.p, b.q, b.p);
e3 cb }

4a 8e bool has_intersection(vector<line> v) {
7f 57 auto intersects = [&](pair<line, int> a, pair<line, int> b) {
71 a0 return interseg(a.first, b.first);
fa 21 };
33 e1 vector<pair<pt, pair<int, int>>> w;
6d f1 for (int i = 0; i < v.size(); i++) {
07 87 if (v[i].q < v[i].p) swap(v[i].p, v[i].q);
83 e1 w.push_back({v[i].p, {0, i}});
14 03 w.push_back({v[i].q, {1, i}});
9e cb }
1e d1 sort(w.begin(), w.end());
47 7f set<pair<line, int>> se;
28 e5 for (auto i : w) {
2d bf line at = v[i].second.second;
07 29 if (i.second.first == 0) {
a2 14 auto nxt = se.lower_bound({at, i.second.second});
3f d1 if (nxt != se.end() and intersects(*nxt, {at,
    i.second.second})) return 1;
81 25 if (nxt != se.begin() and intersects*(--nxt), {at,
    i.second.second})) return 1;
88 78 se.insert({at, i.second.second});
3e 9d } else {
68 88 auto nxt = se.upper_bound({at, i.second.second}), cur
    = nxt, prev = --cur;
1e b6 if (nxt != se.end() and prev != se.begin()
71 4f and intersects(*nxt, *(--prev))) return 1;
ce cc se.erase(cur);
53 cb }
1c cb }
ad bb return 0;
39 cb }
```

## 2.31 Sequencia de de Bruijn

```
// Se passar sem o terceiro parametro, gera um vetor com valores
// em [0, k) de tamanho k^n de forma que todos os subarrays ciclicos
// de tamanho n ocorrem exatamente uma vez
// Se passar com um limite lim, gera o menor vetor com valores
// em [0, k) que possui lim subarrays de tamanho n distintos
// (assume que lim <= k^n)
//
// Linear no tamanho da resposta
// 19720c
```

```
86 86 vector<int> de_bruijn(int n, int k, int lim = INF) {
fe b5 if (k == 1) return vector<int>(lim == INF ? 1 : n, 0);
d0 5f vector<int> l = {0}, ret; // l eh lyndon word
69 66 while (true) {
00 c8 if (l.size() == 0) {
4b 1b if (lim == INF) break;
be da l.push_back(0);
3f cb }
6c 68 if (n % l.size() == 0) for (int i : l) {
a6 72 ret.push_back(i);
52 c9 if (ret.size() == n+lim-1) return ret;
40 cb }
3e 63 int p = l.size();
58 90 while (l.size() < n) l.push_back(l[l.size()%p]);
7f e7 while (l.size() and l.back() == k-1) l.pop_back();
13 88 if (l.size()) l.back()++;
53 cb }
89 ed return ret;
19 cb }
```

## 2.32 Shortest Addition Chain

```
// Computa o menor numero de adicoes para construir
// cada valor, começando com 1 (e podendo salvar variaveis)
// Retorna um par com a dp e o pai na arvore
// A arvore eh tao que o tamanho da raiz (1) ate x
// contem os valores que devem ser criados para gerar x
// A profundidade de x na arvore eh dp[x]
// DP funciona para ateh 300, mas a arvore soh funciona
// para ateh 148
//
// 84fcff

// recuperacao certa soh ateh 148 (erra para 149, 233, 298)
3d 3d pair<vector<int>, vector<int>> addition_chain() {
a8 16 int MAX = 301;
b3 87 vector<int> dp(MAX), p(MAX);
```

```

8f 1a   for (int n = 2; n < MAX; n++) {
6b 7c       pair<int, int> val = {INF, -1};
53 21       for (int i = 1; i < n; i++) for (int j = i; j < n; j = p[j])
d1 94           if (j == n-i) val = min(val, pair(dp[i]+1, i));
3c eb       tie(dp[n], p[n]) = val;
2f ef       if (n == 9) p[n] = 8;
db ba       if (n == 149 or n == 233) dp[n]--;
a0 cb   }
71 71   return {dp, p};
84 cb }

```

## 2.33 Simple Polygon

```

// Verifica se um poligono com n pontos eh simples
//
// O(n log n)
// c724a4

6e 6e bool operator < (const line& a, const line& b) { // comparador
    pro sweepline
26 19     if (a.p == b.p) return ccw(a.p, a.q, b.q);
3b 23     if (!eq(a.p.x, a.q.x) and (eq(b.p.x, b.q.x) or a.p.x+eps <
        b.p.x))
b8 78         return ccw(a.p, a.q, b.p);
b8 dc     return ccw(a.p, b.q, b.p);
e3 cb }

b6 6f bool simple(vector<pt> v) {
0a 57     auto intersects = [&](pair<line, int> a, pair<line, int> b) {
3d e7         if ((a.second+1)%v.size() == b.second or
37 80             (b.second+1)%v.size() == a.second) return false;
39 a0         return interseg(a.first, b.first);
21 21     };
24 41     vector<line> seg;
28 e1     vector<pair<pt, pair<int, int>>> w;
57 f1     for (int i = 0; i < v.size(); i++) {
34 0a         pt at = v[i], nxt = v[(i+1)%v.size()];
05 82         if (nxt < at) swap(at, nxt);
21 93         seg.push_back(line(at, nxt));
99 f7         w.push_back({at, {0, i}});
ec 69         w.push_back({nxt, {1, i}});
        // casos degenerados estranhos
08 ae         if (isinseg(v[(i+2)%v.size()], line(at, nxt))) return 0;
b2 88         if (isinseg(v[(i+v.size()-1)%v.size()], line(at, nxt)))
            return 0;
1f cb     }
82 d1     sort(w.begin(), w.end());

```

```

52 7f     set<pair<line, int>> se;
75 e5     for (auto i : w) {
f6 ff         line at = seg[i.second.second];
88 29         if (i.second.first == 0) {
7d 14             auto nxt = se.lower_bound({at, i.second.second});
03 7c             if (nxt != se.end() and intersects(*nxt, {at,
                i.second.second})) return 0;
03 b3             if (nxt != se.begin() and intersects(*(--nxt), {at,
                i.second.second})) return 0;
c1 78             se.insert({at, i.second.second});
a3 9d         } else {
8b 88             auto nxt = se.upper_bound({at, i.second.second}), cur
                = nxt, prev = --cur;
07 b6             if (nxt != se.end() and prev != se.begin()
e6 40                 and intersects(*nxt, *(--prev))) return 0;
78 cc             se.erase(cur);
ae cb         }
0f cb     }
e2 6a     return 1;
c7 cb }

```

## 2.34 Sweep Direction

```

// Passa por todas as ordenacoes dos pontos definitas por "direcoes"
// Assume que nao existem pontos coincidentes
//
// O(n^2 log n)
// 6bb68d

4b 4b void sweep_direction(vector<pt> v) {
c5 3d     int n = v.size();
34 16     sort(v.begin(), v.end(), [](pt a, pt b) {
f5 3a         if (a.x != b.x) return a.x < b.x;
d3 57         return a.y > b.y;
ab c0     });
56 b8     vector<int> at(n);
12 51     iota(at.begin(), at.end(), 0);
0e b7     vector<pair<int, int>> swapp;
12 25     for (int i = 0; i < n; i++) for (int j = i+1; j < n; j++)
42 95         swapp.push_back({i, j}), swapp.push_back({j, i});

3c 26     sort(swapp.begin(), swapp.end(), [&](auto a, auto b) {
15 13         pt A = rotate90(v[a.first] - v[a.second]);
ea 24         pt B = rotate90(v[b.first] - v[b.second]);
65 61         if (quad(A) == quad(B) and !sarea2(pt(0, 0), A, B)) return
            a < b;
a9 22         return compare_angle(A, B);

```

```

67 c0 });
c0 4e for (auto par : swapp) {
c8 e2     assert(abs(at[par.first] - at[par.second]) == 1);
a2 a9     int l = min(at[par.first], at[par.second]),
2c 0d         r = n-1 - max(at[par.first], at[par.second]);
        // l e r sao quantos caras tem de cada lado do par de
        // pontos
        // (cada par eh visitado duas vezes)
d9 9c     swap(v[at[par.first]], v[at[par.second]]);
07 1c     swap(at[par.first], at[par.second]);
1e cb }
6b cb }

```

## 2.35 Triangulacao de Delaunay

```

// Computa a triangulacao de Delaunay, o dual
// do diagrama de Voronoi (a menos de casos degenerados)
// Retorna um grafo indexado pelos indices dos pontos, e as arestas
// sao as arestas da triangulacao
// As arestas partindo de um vertice ja vem ordenadas por angulo,
// ou seja, se o vertice v nao esta no convex hull, (v, v_i, v_{i+1})
// eh um triangulo da triangulacao, em que v_i eh o i-esimo vizinho
// Usa o alg d&c, precisa representar MAX_COOR^4, por isso __int128
// pra aguentar valores ateh 1e9
//
// Propriedades:
// 1 - O grafo tem no max 3n-6 arestas
// 2 - Para todo triangulo, a circunf. que passa pelos 3 pontos
//     nao contem estritamente nenhum ponto
// 3 - A MST euclidiana eh subgrafo desse grafo
// 4 - Cada ponto eh vizinho do ponto mais proximo dele
//
// O(n log n)
// 362c83

2a 2a typedef struct QuadEdge* Q;
8d ba struct QuadEdge {
7a 53     int id;
47 11     pt o;
16 41     Q rot, nxt;
16 3e     bool used;

cf 3f     QuadEdge(int id_ = -1, pt o_ = pt(INF, INF)) :
c3 4b         id(id_), o(o_), rot(nullptr), nxt(nullptr), used(false) {}

74 00     Q rev() const { return rot->rot; }
f0 c3     Q next() const { return nxt; }

```

```

51 18     Q prev() const { return rot->next()->rot; }
fb 0d     pt dest() const { return rev()->o; }
eb 21 };

fb 91 Q edge(pt from, pt to, int id_from, int id_to) {
d9 c6     Q e1 = new QuadEdge(id_from, from);
e1 61     Q e2 = new QuadEdge(id_to, to);
73 8f     Q e3 = new QuadEdge;
22 5c     Q e4 = new QuadEdge;
25 e6     tie(e1->rot, e2->rot, e3->rot, e4->rot) = {e3, e4, e2, e1};
01 f2     tie(e1->nxt, e2->nxt, e3->nxt, e4->nxt) = {e1, e2, e4, e3};
4c 1a     return e1;
eb cb }

c6 d8 void splice(Q a, Q b) {
c7 a6     swap(a->nxt->rot->nxt, b->nxt->rot->nxt);
36 da     swap(a->nxt, b->nxt);
6c cb }

75 16 void del_edge(Q& e, Q ne) { // delete e and assign e <- ne
58 cc     splice(e, e->prev());
b2 ee     splice(e->rev(), e->rev()->prev());
66 7e     delete e->rev()->rot, delete e->rev();
0b 52     delete e->rot; delete e;
d3 6b     e = ne;
17 cb }

d5 d0 Q conn(Q a, Q b) {
fa cc     Q e = edge(a->dest(), b->o, a->rev()->id, b->id);
28 f2     splice(e, a->rev()->prev());
71 d3     splice(e->rev(), b);
c6 6b     return e;
c5 cb }

d9 d6 bool in_c(pt a, pt b, pt c, pt p) { // p ta na circunf. (a, b,
c) ?
df 26     __int128 p2 = p*p, A = a*a - p2, B = b*b - p2, C = c*c - p2;
3f cb     return sarea2(p, a, b) * C + sarea2(p, b, c) * A + sarea2(p,
c, a) * B > 0;
db cb }

4a 54 pair<Q, Q> build_tr(vector<pt>& p, int l, int r) {
67 09     if (r-l+1 <= 3) {
08 2e         Q a = edge(p[l], p[l+1], l, l+1), b = edge(p[l+1], p[r],
l+1, r);
f0 91         if (r-l+1 == 2) return {a, a->rev()};
0a 0e         splice(a->rev(), b);

```

```

fa c3      ll ar = sarea2(p[l], p[l+1], p[r]);
a2 1a      Q c = ar ? conn(b, a) : 0;
ed 02      if (ar >= 0) return {a, b->rev()};
25 9d      return {c->rev(), c};
94 cb      }
41 ee      int m = (l+r)/2;
1d 32      auto [la, ra] = build_tr(p, l, m);
fe b9      auto [lb, rb] = build_tr(p, m+1, r);
7c 66      while (true) {
40 b9          if (ccw(lb->o, ra->o, ra->dest())) ra = ra->rev()->prev();
18 45          else if (ccw(lb->o, ra->o, lb->dest())) lb =
lb->rev()->next();
ee f9          else break;
a4 cb      }
1b ca      Q b = conn(lb->rev(), ra);
b4 71      auto valid = [&](Q e) { return ccw(e->dest(), b->dest(),
b->o); };
aa ee      if (ra->o == la->o) la = b->rev();
c0 63      if (lb->o == rb->o) rb = b;
4b 66      while (true) {
be 71          Q L = b->rev()->next();
9a d1          if (valid(L)) while (in_c(b->dest(), b->o, L->dest(),
L->next()->dest()))
31 1c              del_edge(L, L->next());
a6 c7          Q R = b->prev();
3d 2b          if (valid(R)) while (in_c(b->dest(), b->o, R->dest(),
R->prev()->dest()))
af 54              del_edge(R, R->prev());
1c a3          if (!valid(L) and !valid(R)) break;
83 cc          if (!valid(L) or (valid(R) and in_c(L->dest(), L->o, R->o,
R->dest()))))
45 36              b = conn(R, b->rev());
c5 66              else b = conn(b->rev(), L->rev());
5b cb      }
78 a2      return {la, rb};
8b cb      }

d7 b5 vector<vector<int>> delaunay(vector<pt> v) {
10 3d      int n = v.size();
6e 39      auto tmp = v;
9c 13      vector<int> idx(n);
6c 29      iota(idx.begin(), idx.end(), 0);
08 fe      sort(idx.begin(), idx.end(), [&](int l, int r) { return v[l] <
v[r]; });
32 5d      for (int i = 0; i < n; i++) v[i] = tmp[idx[i]];
25 78      assert(unique(v.begin(), v.end()) == v.end());
8d 4a      vector<vector<int>> g(n);

```

```

08 4e      bool col = true;
23 a9      for (int i = 2; i < n; i++) if (sarea2(v[i], v[i-1], v[i-2]))
col = false;
03 bf      if (col) {
ad aa          for (int i = 1; i < n; i++)
3a 83              g[idx[i-1]].push_back(idx[i]),
g[idx[i]].push_back(idx[i-1]);
7c 96          return g;
2f cb      }
29 d3      Q e = build_tr(v, 0, n-1).first;
ce 11      vector<Q> edg = {e};
c8 5d      for (int i = 0; i < edg.size(); e = edg[i++]) {
d2 3e          for (Q at = e; !at->used; at = at->next()) {
77 60              at->used = true;
a1 cf              g[idx[at->id]].push_back(idx[at->rev()->id]);
50 15              edg.push_back(at->rev());
90 cb          }
d3 cb      }
04 96      return g;
36 cb      }

```

## 2.36 Triangulos em Grafos

```

// get_triangles(i) encontra todos os triangulos ijk no grafo
// Custo nas arestas
// retorna {custo do triangulo, {j, k}}
//
// O(m sqrt(m) log(n)) se chamar para todos os vertices
// fladbc

c0 c0 vector<pair<int, int>> g[MAX]; // {para, peso}

c0 d4 #warning o 'g' deve estar ordenado
88 9a vector<pair<int, pair<int, int>>> get_triangles(int i) {
7b 77      vector<pair<int, pair<int, int>>> tri;
77 b2      for (pair<int, int> j : g[i]) {
bc 2b          int a = i, b = j.first;
15 6d          if (g[a].size() > g[b].size()) swap(a, b);
38 eb          for (pair<int, int> c : g[a]) if (c.first != b and c.first
> j.first) {
5f 52              auto it = lower_bound(g[b].begin(), g[b].end(),
make_pair(c.first, -INF));
d7 f5              if (it == g[b].end() or it->first != c.first) continue;
44 0a              tri.push_back({j.second+c.second+it->second, {a == i ?
b : a, c.first}});
4e cb          }
c7 cb      }

```

```
66 f5     return tri;
f1 cb }
```

## 3 Strings

### 3.1 Aho-corasick

```
// query retorna o somatorio do numero de matches de
// todas as stringuinhas na stringona
//
// insert - O(|s| log(SIGMA))
// build - O(N), onde N = somatorio dos tamanhos das strings
// query - O(|s|)
// a30d6e

ea ea namespace aho {
9e 80     map<char, int> to[MAX];
96 c8     int link[MAX], idx, term[MAX], exit[MAX], sobe[MAX];

eb bf     void insert(string& s) {
2c 05         int at = 0;
b6 b4         for (char c : s) {
fd b6             auto it = to[at].find(c);
f8 1c             if (it == to[at].end()) at = to[at][c] = ++idx;
de 36             else at = it->second;
1e cb         }
8e 14         term[at]++, sobe[at]++;
49 cb     }

49 d4 #warning nao esquece de chamar build() depois de inserir
e9 0a     void build() {
52 26         queue<int> q;
30 53         q.push(0);
7e df         link[0] = exit[0] = -1;
93 40         while (q.size()) {
72 37             int i = q.front(); q.pop();
32 3c             for (auto [c, j] : to[i]) {
3b 5d                 int l = link[i];
5a 10                 while (l != -1 and !to[l].count(c)) l = link[l];
62 7a                 link[j] = l == -1 ? 0 : to[l][c];
d5 3a                 exit[j] = term[link[j]] ? link[j] : exit[link[j]];
a3 6f                 if (exit[j]+1) sobe[j] += sobe[exit[j]];
a0 11                 q.push(j);
56 cb             }
1d cb         }
2e cb     }

63 bc     int query(string& s) {
```

```
0c 86         int at = 0, ans = 0;
98 b4         for (char c : s){
34 1c             while (at != -1 and !to[at].count(c)) at = link[at];
e6 5b             at = at == -1 ? 0 : to[at][c];
7e 2b             ans += sobe[at];
05 cb         }
ff ba         return ans;
b9 cb     }
a3 cb }
```

### 3.2 Algoritmo Z

```
// z[i] = lcp(s, s[i..n))
//
// Complexidades:
// z - O(|s|)
// match - O(|s| + |p|)
// 74a9e1

a1 a1 vector<int> get_z(string s) {
08 16     int n = s.size();
f5 2b     vector<int> z(n, 0);

c0 fa     int l = 0, r = 0;
0e 6f     for (int i = 1; i < n; i++) {
87 0a         if (i <= r) z[i] = min(r - i + 1, z[i - 1]);
33 45         while (i + z[i] < n and s[z[i]] == s[i + z[i]]) z[i]++;
48 65         if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
39 cb     }

41 07     return z;
74 cb }
```

### 3.3 Automato de Sufixo

```
// Automato que aceita os sufixos de uma string
// Todas as funcoes sao lineares
// c37a72

16 16 namespace sam {
e3 c1     int cur, sz, len[2*MAX], link[2*MAX], acc[2*MAX];
fe 0b     int nxt[2*MAX][26];

14 e6     void add(int c) {
16 17         int at = cur;
18 9a         len[sz] = len[cur]+1, cur = sz++;
```

```

8d 50     while (at != -1 and !nxt[at][c]) nxt[at][c] = cur, at =
link[at];
9b 7e     if (at == -1) { link[cur] = 0; return; }
28 65     int q = nxt[at][c];
2a fd     if (len[q] == len[at]+1) { link[cur] = q; return; }
06 31     int qq = sz++;
fa 2c     len[qq] = len[at]+1, link[qq] = link[q];
ac 9a     for (int i = 0; i < 26; i++) nxt[qq][i] = nxt[q][i];
02 e7     while (at != -1 and nxt[at][c] == q) nxt[at][c] = qq, at =
link[at];
85 8b     link[cur] = link[q] = qq;
e7 cb     }
57 94     void build(string& s) {
05 88         cur = 0, sz = 0, len[0] = 0, link[0] = -1, sz++;
3d 9f         for (auto i : s) add(i-'a');
35 17         int at = cur;
6e 12         while (at) acc[at] = 1, at = link[at];
64 cb     }

// coisas que da pra fazer:
15 28     ll distinct_substrings() {
b8 04         ll ans = 0;
1e a1         for (int i = 1; i < sz; i++) ans += len[i] - len[link[i]];
70 ba         return ans;
f1 cb     }
d2 a6     string longest_common_substring(string& S, string& T) {
e7 41         build(S);
ca 11         int at = 0, l = 0, ans = 0, pos = -1;
34 d5         for (int i = 0; i < T.size(); i++) {
fe f2             while (at and !nxt[at][T[i]-'a']) at = link[at], l =
len[at];
ac ef             if (nxt[at][T[i]-'a']) at = nxt[at][T[i]-'a'], l++;
52 74             else at = 0, l = 0;
64 a1             if (l > ans) ans = l, pos = i;
47 cb         }
f2 20         return T.substr(pos-ans+1, ans);
d2 cb     }
d4 46     ll dp[2*MAX];
20 45     ll paths(int i) {
a8 2a         auto& x = dp[i];
35 de         if (x) return x;
a7 48         x = 1;
e0 71         for (int j = 0; j < 26; j++) if (nxt[i][j]) x +=
paths(nxt[i][j]);
18 ea         return x;
1f cb     }
ba 10     void kth_substring(int k, int at=0) { // k=1 : menor substring

```

```

lexicog.
fd 9d     for (int i = 0; i < 26; i++) if (k and nxt[at][i]) {
e7 d5         if (paths(nxt[at][i]) >= k) {
7e d0             cout << char('a'+i);
e9 c4             kth_substring(k-1, nxt[at][i]);
50 50             return;
3e cb         }
44 5f         k -= paths(nxt[at][i]);
97 cb     }
08 cb     }
c3 21 };

```

### 3.4 eertree

```

// Constroi a eertree, caractere a caractere
// Inicializar com a quantidade de caracteres maxima
// size() retorna a quantidade de substrings pal. distintas
// depois de chamar propagate(), cada substring palindromica
// ocorre qt[i] vezes. O propagate() retorna o numero de
// substrings pal. com repeticao
//
// O(n) amortizado, considerando alfabeto O(1)
// a2e693

8e 8e     struct eertree {
f8 7c         vector<vector<int>> t;
b6 42         int n, last, sz;
f6 74         vector<int> s, len, link, qt;

f7 d3         eertree(int N) {
81 ec             t = vector(N+2, vector(26, int()));
fe ce             s = len = link = qt = vector<int>(N+2);
6c cd             s[0] = -1;
9b 28             link[0] = 1, len[0] = 0, link[1] = 1, len[1] = -1;
ee 68             sz = 2, last = 0, n = 1;
2e cb         }

f7 24         void add(char c) {
a7 69             s[n++] = c - 'a';
e1 34             while (s[n-len[last]-2] != c) last = link[last];
ff 28             if (!t[last][c]) {
af da                 int prev = link[last];
6f 55                 while (s[n-len[prev]-2] != c) prev = link[prev];
fd fb                 link[sz] = t[prev][c];
93 3f                 len[sz] = len[last]+2;
f1 1f                 t[last][c] = sz++;
96 cb             }

```

```

ca 34      qt[last = t[last][c]]++;
4a cb      }
b3 f1      int size() { return sz-2; }
ff 2a      ll propagate() {
e9 b7          ll ret = 0;
9d eb          for (int i = n; i > 1; i--) {
85 fd              qt[link[i]] += qt[i];
1f db              ret += qt[i];
be cb          }
18 ed          return ret;
d4 cb      }
a2 21 };

```

### 3.5 KMP

```

// matching(s, t) retorna os indices das ocorrencias
// de s em t
// autKMP constroi o automato do KMP
//
// Complexidades:
// pi - O(n)
// match - O(n + m)
// construir o automato - O(|sigma|*n)
// n = |padrao| e m = |texto|

// f50359
ea ea template<typename T> vector<int> pi(T s) {
4c 01      vector<int> p(s.size());
20 72      for (int i = 1, j = 0; i < s.size(); i++) {
61 a5          while (j and s[j] != s[i]) j = p[j-1];
ec 97          if (s[j] == s[i]) j++;
34 f8          p[i] = j;
4d cb      }
d1 74      return p;
f5 cb }

// c82524
f3 c1 template<typename T> vector<int> matching(T& s, T& t) {
b9 65      vector<int> p = pi(s), match;
02 a1      for (int i = 0, j = 0; i < t.size(); i++) {
0e 6b          while (j and s[j] != t[i]) j = p[j-1];
53 c4          if (s[j] == t[i]) j++;
7b 31          if (j == s.size()) match.push_back(i-j+1), j = p[j-1];
f9 cb      }
d4 ed      return match;
44 cb }

```

```

// 79bd9e
2c a2 struct KMPaut : vector<vector<int>>> {
1d 47      KMPaut(){}
51 6c      KMPaut(string& s) : vector<vector<int>>>(26,
          vector<int>(s.size()+1)) {
78 50          vector<int> p = pi(s);
39 04          auto& aut = *this;
d1 4f          aut[s[0]-'a'][0] = 1;
12 19          for (char c = 0; c < 26; c++)
7b 5d              for (int i = 1; i <= s.size(); i++)
33 42                  aut[c][i] = s[i]-'a' == c ? i+1 : aut[c][p[i-1]];
4f cb      }
ff 21 };

```

### 3.6 Manacher

```

// manacher recebe um vetor de T e retorna o vetor com tamanho dos
// palindromos
// ret[2*i] = tamanho do maior palindromo centrado em i
// ret[2*i+1] = tamanho maior palindromo centrado em i e i+1
//
// Complexidades:
// manacher - O(n)
// palindrome - <O(n), O(1)>
// pal_end - O(n)

// ebb184
28 28 template<typename T> vector<int> manacher(const T& s) {
07 18      int l = 0, r = -1, n = s.size();
61 fc      vector<int> d1(n), d2(n);
6d 60      for (int i = 0; i < n; i++) {
d3 82          int k = i > r ? 1 : min(d1[l+r-i], r-i);
d8 61          while (i+k < n && i-k >= 0 && s[i+k] == s[i-k]) k++;
94 61          d1[i] = k--;
59 9f          if (i+k > r) l = i-k, r = i+k;
63 cb      }
30 e0      l = 0, r = -1;
d8 60      for (int i = 0; i < n; i++) {
d4 a6          int k = i > r ? 0 : min(d2[l+r-i+1], r-i+1); k++;
f7 2c          while (i+k <= n && i-k >= 0 && s[i+k-1] == s[i-k]) k++;
f9 ea          d2[i] = --k;
4d 26          if (i+k-1 > r) l = i-k, r = i+k-1;
bc cb      }
1a c4      vector<int> ret(2*n-1);
bd e6      for (int i = 0; i < n; i++) ret[2*i] = 2*d1[i]-1;
d9 e1      for (int i = 0; i < n-1; i++) ret[2*i+1] = 2*d2[i+1];
cc ed      return ret;

```



```

eb cb }

// 60c6f5
// verifica se a string s[i..j] eh palindromo
b5 ca template<typename T> struct palindrome {
14 f9     vector<int> man;

ec b2     palindrome(const T& s) : man(manacher(s)) {}
35 9d     bool query(int i, int j) {
62 ba         return man[i+j] >= j-i+1;
61 cb     }
12 21 };

// 8bd4d5
// tamanho do maior palindromo que termina em cada posicao
cb 7c template<typename T> vector<int> pal_end(const T& s) {
71 e5     vector<int> ret(s.size());
20 fd     palindrome<T> p(s);
e7 d5     ret[0] = 1;
d5 88     for (int i = 1; i < s.size(); i++) {
95 a3         ret[i] = min(ret[i-1]+2, i+1);
6e 6e         while (!p.query(i-ret[i]+1, i)) ret[i]--;
46 cb     }
4c ed     return ret;
89 cb }

```

### 3.7 Min/max suffix/cyclic shift

```

// Computa o indice do menor/menor sufixo/cyclic shift
// da string, lexicograficamente
//
// O(n)
// af0367

01 01 template<typename T> int max_suffix(T s, bool mi = false) {
61 47     s.push_back(*min_element(s.begin(), s.end())-1);
ba 1a     int ans = 0;
83 88     for (int i = 1; i < s.size(); i++) {
6d ee         int j = 0;
e3 70         while (ans+j < i and s[i+j] == s[ans+j]) j++;
be 7a         if (s[i+j] > s[ans+j]) {
2b b5             if (!mi or i != s.size()-2) ans = i;
bc c0         } else if (j) i += j-1;
0c cb     }
87 ba     return ans;
f2 cb }

```

```

94 a1 template<typename T> int min_suffix(T s) {
36 76     for (auto& i : s) i *= -1;
e9 09     s.push_back(*max_element(s.begin(), s.end())+1);
92 92     return max_suffix(s, true);
95 cb }

```

```

12 97 template<typename T> int max_cyclic_shift(T s) {
3c 16     int n = s.size();
48 1a     for (int i = 0; i < n; i++) s.push_back(s[i]);
fd 20     return max_suffix(s);
9d cb }

```

```

16 08 template<typename T> int min_cyclic_shift(T s) {
5c 76     for (auto& i : s) i *= -1;
75 7b     return max_cyclic_shift(s);
af cb }

```

### 3.8 String Hashing

```

// Complexidades:
// construtor - O(|s|)
// operator() - O(1)
// 918dfb

87 87 mt19937 rng((int)
    chrono::steady_clock::now().time_since_epoch().count());

8a 46 int uniform(int l, int r) {
d7 a7     uniform_int_distribution<int> uid(l, r);
5c f5     return uid(rng);
c2 cb }

c8 9e template<int MOD> struct str_hash { // 116fcb
12 c6     static int P;
7e dc     vector<ll> h, p;
ee ea     str_hash(string s) : h(s.size()), p(s.size()) {
62 7a         p[0] = 1, h[0] = s[0];
76 ad         for (int i = 1; i < s.size(); i++)
ec 84             p[i] = p[i-1]*P%MOD, h[i] = (h[i-1]*P + s[i])%MOD;
15 cb     }
51 af     ll operator()(int l, int r) { // retorna hash s[l...r]
8d 74         ll hash = h[r] - (l ? h[l-1]*p[r-l+1]%MOD : 0);
d8 df         return hash < 0 ? hash + MOD : hash;
26 cb     }
69 21 };
91 21 template<int MOD> int str_hash<MOD>::P = uniform(256, MOD - 1);
// 1 > |sigma|

```



### 3.9 String Hashing - modulo $2^{61} - 1$

```
// Quase duas vezes mais lento
//
// Complexidades:
// build - O(|s|)
// operator() - O(1)
//
// d3c0f0

9d 9d const ll MOD = (1ll<<61) - 1;
b7 e3 ll mulmod(ll a, ll b) {
7c ff     const static ll LOWER = (1ll<<30) - 1, GET31 = (1ll<<31) - 1;
24 41     ll l1 = a&LOWER, h1 = a>>30, l2 = b&LOWER, h2 = b>>30;
c5 d5     ll m = l1*h2 + l2*h1, h = h1*h2;
43 78     ll ans = l1*l2 + (h>>1) + ((h&1)<<60) + (m>>31) +
        ((m&GET31)<<30) + 1;
a1 1d     ans = (ans&MOD) + (ans>>61), ans = (ans&MOD) + (ans>>61);
71 c0     return ans - 1;
b7 cb }

1c 79 mt19937_64
        rng(chrono::steady_clock::now().time_since_epoch().count());

fb f8 ll uniform(ll l, ll r) {
1e 96     uniform_int_distribution<ll> uid(l, r);
7c f5     return uid(rng);
ad cb }

d8 d7 struct str_hash {
71 c2     static ll P;
49 dc     vector<ll> h, p;
3f ea     str_hash(string s) : h(s.size()), p(s.size()) {
7e 7a         p[0] = 1, h[0] = s[0];
46 ad         for (int i = 1; i < s.size(); i++)
20 63             p[i] = mulmod(p[i - 1], P), h[i] = (mulmod(h[i - 1],
        P) + s[i])%MOD;
1d cb     }
4c af     ll operator()(int l, int r) { // retorna hash s[l...r]
7e 53         ll hash = h[r] - (l ? mulmod(h[l - 1], p[r - l + 1]) : 0);
71 df         return hash < 0 ? hash + MOD : hash;
e1 cb     }
21 21 };
d3 6c ll str_hash::P = uniform(256, MOD - 1); // 1 > |sigma|
```

### 3.10 Suffix Array - $O(n \log n)$

```
// kasai recebe o suffix array e calcula lcp[i],
// o lcp entre s[sa[i],...,n-1] e s[sa[i+1],...,n-1]
//
// Complexidades:
// suffix_array - O(n log(n))
// kasai - O(n)
// d3a6ce

73 73 vector<int> suffix_array(string s) {
d3 b3     s += "$";
04 04     int n = s.size(), N = max(n, 260);
d1 2f     vector<int> sa(n), ra(n);
eb 29     for(int i = 0; i < n; i++) sa[i] = i, ra[i] = s[i];

c9 0a     for(int k = 0; k < n; k ? k *= 2 : k++) {
e1 5c         vector<int> nsa(sa), nra(n), cnt(N);

2a fa             for(int i = 0; i < n; i++) nsa[i] = (nsa[i]-k+n)%n,
                    cnt[ra[i]]++;
64 4c             for(int i = 1; i < N; i++) cnt[i] += cnt[i-1];
07 36             for(int i = n-1; i+1; i--) sa[--cnt[ra[nsa[i]]]] = nsa[i];

47 28             for(int i = 1, r = 0; i < n; i++) nra[sa[i]] = r +=
                    ra[sa[i]] !=
44 f8                 ra[sa[i-1]] or ra[(sa[i]+k)%n] != ra[(sa[i-1]+k)%n];
b5 26                 ra = nra;
9a d5                 if (ra[sa[n-1]] == n-1) break;
70 cb             }
e8 05     return vector<int>(sa.begin()+1, sa.end());
ff cb }

fc 48 vector<int> kasai(string s, vector<int> sa) {
24 23     int n = s.size(), k = 0;
e9 40     vector<int> ra(n), lcp(n);
3f 67     for (int i = 0; i < n; i++) ra[sa[i]] = i;

40 74     for (int i = 0; i < n; i++, k -= !!k) {
6f 19         if (ra[i] == n-1) { k = 0; continue; }
0c 1d         int j = sa[ra[i]+1];
16 89         while (i+k < n and j+k < n and s[i+k] == s[j+k]) k++;
bb d9         lcp[ra[i]] = k;
2a cb     }
51 5e     return lcp;
d3 cb }
```

### 3.11 Suffix Array - $O(n)$

```

// Rapidao
// Computa o suffix array em 'sa', o rank em 'rnk'
// e o lcp em 'lcp'
// query(i, j) retorna o LCP entre s[i..n-1] e s[j..n-1]
//
// Complexidades
// O(n) para construir
// query - O(1)

// hash do arquivo inteiro: fa533e

// bab412
1a 1a template<typename T> struct rmq {
9e 51     vector<T> v;
4b fc     int n; static const int b = 30;
52 70     vector<int> mask, t;

4e 18     int op(int x, int y) { return v[x] <= v[y] ? x : y; }
a9 ee     int msb(int x) { return __builtin_clz(1)-__builtin_clz(x); }
a1 c9     int small(int r, int sz = b) { return
        r-msb(mask[r]&((1<<sz)-1)); }
ed 6a     rmq() {}
e7 43     rmq(const vector<T>& v_) : v(v_), n(v.size()), mask(n), t(n) {
0e 2e         for (int i = 0, at = 0; i < n; mask[i++] = at |= 1) {
4c a6             at = (at<<1)&((1<<b)-1);
87 c0             while (at and op(i-msb(at&-at), i) == i) at ^= at&-at;
47 cb         }
8e ea         for (int i = 0; i < n/b; i++) t[i] = small(b*i+b-1);
5b 39         for (int j = 1; (1<<j) <= n/b; j++) for (int i = 0;
            i+(1<<j) <= n/b; i++)
c7 ba             t[n/b*j+i] = op(t[n/b*(j-1)+i],
            t[n/b*(j-1)+i+(1<<(j-1))]);
cf cb     }
e2 e3     int index_query(int l, int r) {
dc 27         if (r-l+1 <= b) return small(r, r-l+1);
f8 e8         int x = l/b+1, y = r/b-1;
d6 fd         if (x > y) return op(small(l+b-1), small(r));
97 a4         int j = msb(y-x+1);
b6 ea         int ans = op(small(l+b-1), op(t[n/b*j+x],
            t[n/b*j+y-(1<<j)+1]));
1e be         return op(ans, small(r));
d4 cb     }
30 09     T query(int l, int r) { return v[index_query(l, r)]; }
ba 21 };

2b 9d struct suffix_array {
95 ac     string s;

```

```

4c 1a     int n;
2f 5b     vector<int> sa, cnt, rnk, lcp;
c3 2d     rmq<int> RMQ;

a4 d6     bool cmp(int a1, int b1, int a2, int b2, int a3=0, int b3=0) {
64 91         return a1 != b1 ? a1 < b1 : (a2 != b2 ? a2 < b2 : a3 < b3);
d0 cb     }
0b 4a     template<typename T> void radix(int* fr, int* to, T* r, int N,
        int k) {
78 c1         cnt = vector<int>(k+1, 0);
a8 ba         for (int i = 0; i < N; i++) cnt[r[fr[i]]]++;
89 70         for (int i = 1; i <= k; i++) cnt[i] += cnt[i-1];
e8 00         for (int i = N-1; i+1; i--) to[--cnt[r[fr[i]]]] = fr[i];
19 cb     }
f9 d6     void rec(vector<int>& v, int k) {
cb a7         auto &tmp = rnk, &m0 = lcp;
da 3a         int N = v.size()-3, sz = (N+2)/3, sz2 = sz+N/3;
bd 7f         vector<int> R(sz2+3);
1a 74         for (int i = 1, j = 0; j < sz2; i += i%3) R[j++] = i;

49 b3         radix(&R[0], &tmp[0], &v[0]+2, sz2, k);
a4 20         radix(&tmp[0], &R[0], &v[0]+1, sz2, k);
fa 5f         radix(&R[0], &tmp[0], &v[0]+0, sz2, k);

51 af         int dif = 0;
30 ed         int l0 = -1, l1 = -1, l2 = -1;
85 d8         for (int i = 0; i < sz2; i++) {
70 8d             if (v[tmp[i]] != l0 or v[tmp[i]+1] != l1 or
                v[tmp[i]+2] != l2)
81 b4                 l0 = v[tmp[i]], l1 = v[tmp[i]+1], l2 =
                v[tmp[i]+2], dif++;
46 19             if (tmp[i]%3 == 1) R[tmp[i]/3] = dif;
9e 1f             else R[tmp[i]/3+sz] = dif;
e3 cb         }

3a 47         if (dif < sz2) {
4e 14             rec(R, dif);
72 74             for (int i = 0; i < sz2; i++) R[sa[i]] = i+1;
e4 8b         } else for (int i = 0; i < sz2; i++) sa[R[i]-1] = i;

de 6f         for (int i = 0, j = 0; j < sz2; i++) if (sa[i] < sz)
            tmp[j++] = 3*sa[i];
92 7c         radix(&tmp[0], &m0[0], &v[0], sz, k);
db 74         for (int i = 0; i < sz2; i++)
fa c9             sa[i] = sa[i] < sz ? 3*sa[i]+1 : 3*(sa[i]-sz)+2;

4b 33         int at = sz2+sz-1, p = sz-1, p2 = sz2-1;

```

```

b1 1c         while (p >= 0 and p2 >= 0) {
1e 3b             if ((sa[p2]%3==1 and cmp(v[m0[p]], v[sa[p2]]),
                R[m0[p]/3],
2d 0c                 R[sa[p2]/3+sz])) or (sa[p2]%3==2 and cmp(v[m0[p]],
                v[sa[p2]],
3d af                 v[m0[p]+1], v[sa[p2]+1], R[m0[p]/3+sz],
                R[sa[p2]/3+1]))))
04 30                 sa[at--] = sa[p2--];
0e cb             else sa[at--] = m0[p--];
f1 cb         }
ec f2         while (p >= 0) sa[at--] = m0[p--];
a3 eb         if (N%3==1) for (int i = 0; i < N; i++) sa[i] = sa[i+1];
f3 cb     }

77 93     suffix_array(const string& s_) : s(s_), n(s.size()), sa(n+3),
13 e6         cnt(n+1), rnk(n), lcp(n-1) {
49 9f         vector<int> v(n+3);
7d f9         for (int i = 0; i < n; i++) v[i] = i;
02 eb         radix(&v[0], &rnk[0], &s[0], n, 256);
d8 e6         int dif = 1;
8b 83         for (int i = 0; i < n; i++)
d1 41             v[rnk[i]] = dif += (i and s[rnk[i]] != s[rnk[i-1]]);
65 7c         if (n >= 2) rec(v, dif);
6b fb         sa.resize(n);

f6 76         for (int i = 0; i < n; i++) rnk[sa[i]] = i;
41 89         for (int i = 0, k = 0; i < n; i++, k -= !!k) {
4c 66             if (rnk[i] == n-1) {
d3 5a                 k = 0;
e5 5e                 continue;
7f cb             }
a7 39             int j = sa[rnk[i]+1];
8c 89             while (i+k < n and j+k < n and s[i+k] == s[j+k]) k++;
54 82             lcp[rnk[i]] = k;
18 cb         }
26 9f         RMQ = rmq<int>(lcp);
3e cb     }

// hash ateh aqui (sem o RMQ): 1ff700

4c 58     int query(int i, int j) {
33 d9         if (i == j) return n-i;
62 22         i = rnk[i], j = rnk[j];
d5 c3         return RMQ.query(min(i, j), max(i, j)-1);
f1 cb     }

76 71     pair<int, int> next(int L, int R, int i, char c) {
15 02         int l = L, r = R+1;
96 40         while (l < r) {

```

```

63 ee             int m = (l+r)/2;
ac e7             if (i+sa[m] >= n or s[i+sa[m]] < c) l = m+1;
db ef             else r = m;
88 cb         }
dc 57         if (l == R+1 or s[i+sa[l]] > c) return {-1, -1};
1a eb         L = l;

cd 9e         l = L, r = R+1;
d0 40         while (l < r) {
12 ee             int m = (l+r)/2;
4f 1a             if (i+sa[m] >= n or s[i+sa[m]] <= c) l = m+1;
25 ef             else r = m;
ce cb         }
f5 56         R = l-1;
ba e1         return {L, R};
28 cb     }

// quantas vezes 't' ocorre em 's' - O(|t| log n)
40 66     int count_substr(string& t) {
ce b2         int L = 0, R = n-1;
ae c9         for (int i = 0; i < t.size(); i++) {
c6 de             tie(L, R) = next(L, R, i, t[i]);
f3 4f             if (L == -1) return 0;
f5 cb         }
74 fb         return R-L+1;
f1 cb     }

// exemplo de f que resolve o problema
//
https://codeforces.com/edu/course/2/lesson/2/5/practice/contest
00 57     ll f(ll k) { return k*(k+1)/2; }

9c e6     ll dfs(int L, int R, int p) { // dfs na suffix tree chamado em
pre ordem
5e c5         int ext = L != R ? RMQ.query(L, R-1) : n - sa[L];

// Tem 'ext - p' substrings diferentes que ocorrem 'R-L+1'
vezes
// 0 LCP de todas elas eh 'ext'
54 f8         ll ans = (ext-p)*f(R-L+1);

// L eh terminal, e folha sse L == R
fd 63         if (sa[L]+ext == n) L++;

// se for um SA de varias strings separadas como s#t$u&,
usar no lugar do if de cima
// (separadores < 'a', diferentes e inclusive no final)
// while (L <= R && (sa[L]+ext == n || s[sa[L]+ext] <

```

```

        'a')) {
//      L++;
// }

51 ad      while (L <= R) {
09 5a          int idx = L != R ? RMQ.index_query(L, R-1) : -1;
e4 5e          if (idx == -1 or lcp[idx] != ext) idx = R;

0a 47          ans += dfs(L, idx, ext);
16 28          L = idx+1;
f1 cb      }
83 ba      return ans;
a0 cb  }

// sum over substrings: computa, para toda substring t
// distinta de s,
// \sum f(# ocorrencias de t em s) - 0 (n)
3a ca  ll sos() { return dfs(0, n-1, 0); }
fa 21 };

```

### 3.12 Suffix Array Dinamico

```

// Mantem o suffix array, lcp e rank de uma string,
// permitindo push_front e pop_front
// 0 operador [i] return um par com sa[i] e lcp[i]
// lcp[i] tem o lcp entre sa[i] e sa[i-1] (lcp[0] = 0)
//
// Complexidades:
// Construir sobre uma string de tamanho n: O(n log n)
// push_front e pop_front: O(log n) amortizado
// 4c2a2e

2f 2f struct dyn_sa {
1f 3c     struct node {
b7 1d         int sa, lcp;
a7 ed         node *l, *r, *p;
d7 f0         int sz, mi;
3d 17         node(int sa_, int lcp_, node* p_) : sa(sa_), lcp(lcp_),
81 54             l(NULL), r(NULL), p(p_), sz(1), mi(lcp) {}
96 01         void update() {
4e 58             sz = 1, mi = lcp;
f4 bd             if (l) sz += l->sz, mi = min(mi, l->mi);
c3 a5             if (r) sz += r->sz, mi = min(mi, r->mi);
fd cb         }
a2 21     };

64 bb     node* root;

```

```

98 29     vector<ll> tag; // tag of a suffix (reversed id)
82 ac     string s; // reversed

2f cf     dyn_sa() : root(NULL) {}
21 e4     dyn_sa(string s_) : dyn_sa() {
f3 ae         reverse(s_.begin(), s_.end());
ef 51         for (char c : s_) push_front(c);
4a cb     }
27 a8     ~dyn_sa() {
be 60         vector<node*> q = {root};
a7 40         while (q.size()) {
8f e5             node* x = q.back(); q.pop_back();
b7 ee             if (!x) continue;
f0 1c             q.push_back(x->l), q.push_back(x->r);
5d bf             delete x;
e3 cb         }
c9 cb     }

a8 73     int size(node* x) { return x ? x->sz : 0; }
da 08     int mirror(int i) { return s.size()-1 - i; }
cf 58     bool cmp(int i, int j) {
8d a2         if (s[i] != s[j]) return s[i] < s[j];
58 5b         if (i == 0 or j == 0) return i < j;
5a 98         return tag[i-1] < tag[j-1];
1b cb     }
4c 91     void fix_path(node* x) { while (x) x->update(), x = x->p; }
37 24     void flatten(vector<node*>& v, node* x) {
d2 8c         if (!x) return;
d8 e9         flatten(v, x->l);
ff 2a         v.push_back(x);
c0 42         flatten(v, x->r);
5b cb     }
2c 96     void build(vector<node*>& v, node*& x, node* p, int L, int R,
ll 1, ll r) {
f4 04         if (L > R) return void(x = NULL);
02 33         int M = (L+R)/2;
14 3e         ll m = (l+r)/2;
8e 7e         x = v[M];
21 63         x->p = p;
8f bb         tag[x->sa] = m;
2f ae         build(v, x->l, x, L, M-1, l, m-1), build(v, x->r, x, M+1,
R, m+1, r);
04 ca         x->update();
8b cb     }
bc 82     void fix(node*& x, node* p, ll l, ll r) {
f9 7f         if (3*max(size(x->l), size(x->r)) <= 2*size(x)) return
x->update();

```

```

15 3d      vector<node*> v;
c0 0c      flatten(v, x);
fb ea      build(v, x, p, 0, v.size()-1, l, r);
85 cb      }
69 b1      node* next(node* x) {
d2 72          if (x->r) {
a2 a9              x = x->r;
0c 34              while (x->l) x = x->l;
90 ea              return x;
a4 cb          }
85 40          while (x->p and x->p->r == x) x = x->p;
e1 13          return x->p;
0d cb      }
3d b6      node* prev(node* x) {
38 e4          if (x->l) {
09 a2              x = x->l;
68 93              while (x->r) x = x->r;
f1 ea              return x;
b4 cb          }
73 6a          while (x->p and x->p->l == x) x = x->p;
7a 13          return x->p;
dd cb      }

f0 4f      int get_lcp(node* x, node* y) {
88 75          if (!x or !y) return 0; // change default value here
84 e5          if (s[x->sa] != s[y->sa]) return 0;
11 84          if (x->sa == 0 or y->sa == 0) return 1;
41 4d          return 1 + query(mirror(x->sa-1), mirror(y->sa-1));
54 cb      }
c6 ad      void add_suf(node*& x, node* p, int id, ll l, ll r) {
cc 91          if (!x) {
06 8e              x = new node(id, 0, p);
de 8e              node *prv = prev(x), *nxt = next(x);
65 65              int lcp_cur = get_lcp(prv, x), lcp_nxt = get_lcp(x,
nxt);
2f ca              if (nxt) nxt->lcp = lcp_nxt, fix_path(nxt);
df 71              x->lcp = lcp_cur;
a4 7b              tag[id] = (l+r)/2;
77 ca              x->update();
bf 50              return;
62 cb          }
39 4a          if (cmp(id, x->sa)) add_suf(x->l, x, id, l, tag[x->sa]-1);
81 c3          else add_suf(x->r, x, id, tag[x->sa]+1, r);
a2 3d          fix(x, p, l, r);
8c cb      }
c7 ec      void push_front(char c) {
46 cc          s += c;

```

```

35 49      tag.push_back(-1);
04 05      add_suf(root, NULL, s.size() - 1, 0, 1e18);
40 cb      }

ba 7f      void rem_suf(node*& x, int id) {
28 6c          if (x->sa != id) {
c7 86              if (tag[id] < tag[x->sa]) return rem_suf(x->l, id);
62 e6              return rem_suf(x->r, id);
cd cb          }
bd 2c          node* nxt = next(x);
86 09          if (nxt) nxt->lcp = min(nxt->lcp, x->lcp), fix_path(nxt);

1a b2          node *p = x->p, *tmp = x;
f9 f3          if (!x->l or !x->r) {
c7 2f              x = x->l ? x->l : x->r;
72 75              if (x) x->p = p;
5a 9d          } else {
45 7f              for (tmp = x->l, p = x; tmp->r; tmp = tmp->r) p = tmp;
e6 f2              x->sa = tmp->sa, x->lcp = tmp->lcp;
0e 48              if (tmp->l) tmp->l->p = p;
00 14              if (p->l == tmp) p->l = tmp->l;
5e a9              else p->r = tmp->l;
67 cb          }
64 b5          fix_path(p);
1f 7c          delete tmp;
65 cb      }
c1 15      void pop_front() {
c7 ab          if (!s.size()) return;
6b 34          s.pop_back();
0e 43          rem_suf(root, s.size());
fb c6          tag.pop_back();
3a cb      }

d6 53      int query(node* x, ll l, ll r, ll a, ll b) {
04 e5          if (!x or tag[x->sa] == -1 or r < a or b < l) return
s.size();
3f ef          if (a <= l and r <= b) return x->mi;
c8 8e          int ans = s.size();
92 e1          if (a <= tag[x->sa] and tag[x->sa] <= b) ans = min(ans,
x->lcp);
e2 d9          ans = min(ans, query(x->l, l, tag[x->sa]-1, a, b));
df 26          ans = min(ans, query(x->r, tag[x->sa]+1, r, a, b));
bf ba          return ans;
2c cb      }
02 58      int query(int i, int j) { // lcp(s[i..], s[j..])
dd 20          if (i == j) return s.size() - i;
bf 29          ll a = tag[mirror(i)], b = tag[mirror(j)];

```

```

4a 71     int ret = query(root, 0, 1e18, min(a, b)+1, max(a, b));
6d ed     return ret;
db cb }
// optional: get rank[i], sa[i] and lcp[i]
f0 04 int rank(int i) {
3e 39     i = mirror(i);
15 52     node* x = root;
51 7c     int ret = 0;
be f4     while (x) {
1e 33         if (tag[x->sa] < tag[i]) {
6e f9             ret += size(x->l)+1;
3b a9             x = x->r;
d7 eb         } else x = x->l;
ef cb     }
2b ed     return ret;
bc cb }
c0 64 pair<int, int> operator[](int i) {
6d 52     node* x = root;
59 31     while (1) {
83 d4         if (i < size(x->l)) x = x->l;
ca 4e         else {
64 85             i -= size(x->l);
5d e0             if (!i) return {mirror(x->sa), x->lcp};
5f 04             i--, x = x->r;
1c cb         }
1e cb     }
4d cb }
4c 21 };

```

### 3.13 Trie

```

// trie T() constroi uma trie para o alfabeto das letras minusculas
// trie T(tamanho do alfabeto, menor caracter) tambem pode ser usado
//
// T.insert(s) - O(|s|*sigma)
// T.erase(s) - O(|s|)
// T.find(s) retorna a posicao, 0 se nao achar - O(|s|)
// T.count_pref(s) numero de strings que possuem s como prefixo -
// O(|s|)
//
// Nao funciona para string vazia
// 979609

```

```

ab ab struct trie {
fb e1     vector<vector<int>> to;
71 45     vector<int> end, pref;
ff af     int sigma; char norm;

```

```

c7 bb     trie(int sigma_=26, char norm_='a') : sigma(sigma_),
norm(norm_) {
70 58         to = {vector<int>(sigma)};
39 86         end = {0}, pref = {0};
95 cb     }
c2 64 void insert(string s) {
39 c6         int x = 0;
fc 7e         for(auto c : s) {
58 00             int &nxt = to[x][c-norm];
ee dd             if(!nxt) {
e4 0a                 nxt = to.size();
56 52                 to.push_back(vector<int>(sigma));
f3 77                 end.push_back(0), pref.push_back(0);
c4 cb             }
95 82             x = nxt, pref[x]++;
1d cb         }
41 e4         end[x]++;
4a cb     }
29 6b void erase(string s) {
78 c6         int x = 0;
e1 b4         for(char c : s) {
66 00             int &nxt = to[x][c-norm];
18 10             x = nxt, pref[x]--;
36 d8             if(!pref[x]) nxt = 0;
14 cb         }
d0 bf         end[x]--;
95 cb     }
fe ae int find(string s) {
46 c6         int x = 0;
47 7e         for(auto c : s) {
15 2e             x = to[x][c-norm];
c7 a6             if(!x) return 0;
7d cb         }
56 ea         return x;
f4 cb     }
22 83 int count_pref(string s) {
f3 e2         return pref[find(s)];
a2 cb     }
97 21 };

```

## 4 Matematica

### 4.1 2-SAT

```

// solve() retorna um par, o first fala se eh possivel
// atribuir, o second fala se cada variavel eh verdadeira

```

```

//
// 0(|V|+|E|) = 0(#variaveis + #restricoes)
// ef6b3b

13 13 struct sat {
e1 e6     int n, tot;
c1 78     vector<vector<int>> g;
1a 0c     vector<int> vis, comp, id, ans;
ed 4c     stack<int> s;

ab 14     sat() {}
b9 17     sat(int n_) : n(n_), tot(n), g(2*n) {}

ec f3     int dfs(int i, int& t) {
43 cf         int lo = id[i] = t++;
5d ef         s.push(i), vis[i] = 2;
a1 48         for (int j : g[i]) {
34 74             if (!vis[j]) lo = min(lo, dfs(j, t));
7e 99             else if (vis[j] == 2) lo = min(lo, id[j]);
ba cb         }
71 3d         if (lo == id[i]) while (1) {
4a 3c             int u = s.top(); s.pop();
cc 9c             vis[u] = 1, comp[u] = i;
c1 91             if ((u>>1) < n and ans[u>>1] == -1) ans[u>>1] = ~u&1;
0f 2e             if (u == i) break;
d1 cb         }
93 25         return lo;
13 cb     }

f5 74     void add_impl(int x, int y) { // x -> y = !x ou y
b8 26         x = x >= 0 ? 2*x : -2*x-1;
5c 2b         y = y >= 0 ? 2*y : -2*y-1;
1b a1         g[x].push_back(y);
ce 1e         g[y^1].push_back(x^1);
68 cb     }
0a e8     void add_cl(int x, int y) { // x ou y
62 0b         add_impl(~x, y);
ed cb     }
50 48     void add_xor(int x, int y) { // x xor y
99 0b         add_cl(x, y), add_cl(~x, ~y);
4b cb     }
51 97     void add_eq(int x, int y) { // x = y
e3 c8         add_xor(~x, y);
40 cb     }
21 b1     void add_true(int x) { // x = T
d0 18         add_impl(~x, x);
5b cb     }

```

```

c0 d1     void at_most_one(vector<int> v) { // no max um verdadeiro
84 54         g.resize(2*(tot+v.size()));
24 f1         for (int i = 0; i < v.size(); i++) {
6c 8c             add_impl(tot+i, ~v[i]);
1e a8             if (i) {
75 b6                 add_impl(tot+i, tot+i-1);
72 3d                 add_impl(v[i], tot+i-1);
f8 cb             }
90 cb         }
62 25         tot += v.size();
8b cb     }

e2 a8     pair<bool, vector<int>> solve() {
32 27         ans = vector<int>(n, -1);
40 6b         int t = 0;
89 0d         vis = comp = id = vector<int>(2*tot, 0);
a7 53         for (int i = 0; i < 2*tot; i++) if (!vis[i]) dfs(i, t);
2c f8         for (int i = 0; i < tot; i++)
db 4c             if (comp[2*i] == comp[2*i+1]) return {false, {}};
e1 99         return {true, ans};
98 cb     }
ef 21 };

```

## 4.2 Algoritmo de Euclides estendido

```

// Acha x e y tal que ax + by = mdc(a, b) (nao eh unico)
// Assume a, b >= 0
//
// 0(log(min(a, b)))
// 35411d

```

```

2b 2b tuple<ll, ll, ll> ext_gcd(ll a, ll b) {
29 3b     if (!a) return {b, 0, 1};
10 55     auto [g, x, y] = ext_gcd(b%a, a);
6b c5     return {g, y - b/a*x, x};
35 cb }

```

## 4.3 Avaliacao de Interpolacao

```

// Dado 'n' pontos (i, y[i]), i \in [0, n),
// avalia o polinomio de grau n-1 que passa
// por esses pontos em 'x'
// Tudo modular, precisa do mint
//
// 0(n)
// 4fe929

```

```

ee ee mint evaluate_interpolation(int x, vector<mint> y) {
92 80     int n = y.size();

a4 18     vector<mint> sulf(n+1, 1), fat(n, 1), ifat(n);
14 6f     for (int i = n-1; i >= 0; i--) sulf[i] = sulf[i+1] * (x - i);
65 29     for (int i = 1; i < n; i++) fat[i] = fat[i-1] * i;
aa 0d     ifat[n-1] = 1/fat[n-1];
09 3d     for (int i = n-2; i >= 0; i--) ifat[i] = ifat[i+1] * (i + 1);

12 ca     mint pref = 1, ans = 0;
52 5e     for (int i = 0; i < n; pref *= (x - i++)) {
0d 42         mint num = pref * sulf[i+1];

97 b4         mint den = ifat[i] * ifat[n-1 - i];
18 0b         if ((n-1 - i)%2) den *= -1;

6e 03         ans += y[i] * num * den;
53 cb     }
03 ba     return ans;
4f cb }

```

## 4.4 Berlekamp-Massey

```

// guess_kth(s, k) chuta o k-esimo (0-based) termo
// de uma recorrência linear que gera s
// Para uma rec. lin. de ordem x, se passar 2x termos
// vai gerar a certa
// Usar aritmetica modular
//
// O(n^2 log k), em que n = |s|
// 8644e3

b7 b7 template<typename T> T evaluate(vector<T> c, vector<T> s, ll k) {
64 ff     int n = c.size();
b4 9e     assert(c.size() <= s.size());

3a d0     auto mul = [&](const vector<T> &a, const vector<T> &b) {
27 56         vector<T> ret(a.size() + b.size() - 1);
b5 d7         for (int i = 0; i < a.size(); i++) for (int j = 0; j <
            b.size(); j++)
41 cf             ret[i+j] += a[i] * b[j];
38 83         for (int i = ret.size()-1; i >= n; i--) for (int j = n-1;
            j >= 0; j--)
d7 11             ret[i-j-1] += ret[i] * c[j];
e1 16         ret.resize(min<int>(ret.size(), n));
3a ed         return ret;
42 21     };

```

```

80 1a     vector<T> a = n == 1 ? vector<T>({c[0]}) : vector<T>({0, 1}),
        x = {1};
91 95     while (k) {
8e 7f         if (k&1) x = mul(x, a);
32 b2         a = mul(a, a), k >>= 1;
08 cb     }
24 dd     x.resize(n);

7a ce     T ret = 0;
a6 e7     for (int i = 0; i < n; i++) ret += x[i] * s[i];
85 ed     return ret;
7e cb }

f1 19 template<typename T> vector<T> berlekamp_massey(vector<T> s) {
d3 ce     int n = s.size(), l = 0, m = 1;
a8 22     vector<T> b(n), c(n);
84 46     T ld = b[0] = c[0] = 1;
10 62     for (int i = 0; i < n; i++, m++) {
90 79         T d = s[i];
39 ab         for (int j = 1; j <= l; j++) d += c[j] * s[i-j];
36 5f         if (d == 0) continue;
a6 8b         vector<T> temp = c;
c4 36         T coef = d / ld;
1f ba         for (int j = m; j < n; j++) c[j] -= coef * b[j-m];
30 88         if (2 * l <= i) l = i + 1 - l, b = temp, ld = d, m = 0;
78 cb     }
58 90     c.resize(l + 1);
86 84     c.erase(c.begin());
4a 0d     for (T& x : c) x = -x;
8b 80     return c;
8d cb }

d9 2c template<typename T> T guess_kth(const vector<T>& s, ll k) {
a0 cc     auto c = berlekamp_massey(s);
61 96     return evaluate(c, s, k);
86 cb }

```

## 4.5 Binomial Distribution

```

// binom(n, k, p) retorna a probabilidade de k sucessos
// numa binomial(n, p)
// 00d38f

36 36 double logfact[MAX];

46 9e void calc() {

```



```

65 7a    logfact[0] = 0;
14 15    for (int i = 1; i < MAX; i++) logfact[i] = logfact[i-1] +
        log(i);
3f cb }

c1 94 double binom(int n, int k, double p) {
29 27    return exp(logfact[n] - logfact[k] - logfact[n-k] + k * log(p)
        + (n-k) * log(1 - p));
00 cb }

```

## 4.6 Convolucao de GCD / LCM

```

// O(n log(n))

// multiple_transform(a)[i] = \sum_d a[d * i]
// 338be8
bb bb template<typename T> void multiple_transform(vector<T>& v, bool
    inv = false) {
a9 64    vector<int> I(v.size()-1);
7d 84    iota(I.begin(), I.end(), 1);
b3 67    if (inv) reverse(I.begin(), I.end());
bd da    for (int i : I) for (int j = 2; i*j < v.size(); j++)
6f a8        v[i] += (inv ? -1 : 1) * v[i*j];
33 cb }

// gcd_convolution(a, b)[k] = \sum_{gcd(i, j) = k} a_i * b_j
// 984f53
3b fe template<typename T> vector<T> gcd_convolution(vector<T> a,
    vector<T> b) {
21 bd    multiple_transform(a), multiple_transform(b);
9f 79    for (int i = 0; i < a.size(); i++) a[i] *= b[i];
7b de    multiple_transform(a, true);
8d 3f    return a;
1a cb }

// divisor_transform(a)[i] = \sum_{d|i} a[i/d]
// aa74e5
f3 be template<typename T> void divisor_transform(vector<T>& v, bool
    inv = false) {
14 64    vector<int> I(v.size()-1);
43 84    iota(I.begin(), I.end(), 1);
2f 5e    if (!inv) reverse(I.begin(), I.end());
08 da    for (int i : I) for (int j = 2; i*j < v.size(); j++)
b8 14        v[i*j] += (inv ? -1 : 1) * v[i];
bc cb }

// lcm_convolution(a, b)[k] = \sum_{lcm(i, j) = k} a_i * b_j

```

```

// f5acc1
1b b1 template<typename T> vector<T> lcm_convolution(vector<T> a,
    vector<T> b) {
e2 3a    divisor_transform(a), divisor_transform(b);
77 79    for (int i = 0; i < a.size(); i++) a[i] *= b[i];
9c d8    divisor_transform(a, true);
80 3f    return a;
1d cb }

```

## 4.7 Coprime Basis

```

// Dado um conjunto de elementos A constroi uma base B
// de fatores coprimos tal que todo elemento A[i]
// pode ser fatorado como A[i] = \prod B[j]^p_ij
//
// Sendo n o numero de inserts, a complexidade esperada fica
// O(n*(n*loglog(MAX) + log(MAX)^2))
//
// No pior caso, podemos trocar n*loglog(MAX) por
// se MAX <= 1e6 fica 8*n
// se MAX <= 1e9 fica 10*n
// se MAX <= 1e18 fica 16*n
// se MAX <= 1e36 fica 26*n
//
// 6714d3

eb eb template <typename T> struct coprime_basis {
c1 a0    vector<T> basis;

ce 60    coprime_basis() {}
0e 05    coprime_basis(vector<T> v) { for (T i : v) insert(i); }

08 84    void insert(T z) {
12 c3        int n = basis.size();
39 ef        basis.push_back(z);
e7 43        for (int i = n; i < basis.size(); i++) {
5c 21            for (int j = (i != n) ? i+1 : 0; j < basis.size();
                j++) {
86 4c                if (i == j) continue;
c3 02                T &x = basis[i];
1d c9                if (x == 1) {
7e fa                    j = INF;
0b 5e                    continue;
c0 cb                }
1b 54                T &y = basis[j];
3a 3c                T g = gcd(x, y);
41 e1                if (g == 1) continue;

```

```

74 15          y /= g, x /= g;
fe 8c          basis.push_back(g);
7c cb          }
50 cb          }
86 fe          basis.erase(remove(basis.begin(), basis.end(), 1),
    basis.end());
8a cb          }

95 4b  vector<int> factor(T x) {
83 21      vector<int> fat(basis.size());
1f 6f      for (int i = 0; i < basis.size(); i++) {
28 25          while (x % basis[i] == 0) x /= basis[i], fat[i]++;
8b cb      }
75 6a      return fat;
7f cb  }
67 21 };

```

## 4.8 Crivo de Eratosthenes

```

// "0" crivo
//
// Encontra maior divisor primo
// Um numero eh primo sse divi[x] == x
// fact fatora um numero <= lim
// A fatoracao sai ordenada
//
// crivo - O(n log(log(n)))
// fact - O(log(n))

// hash (crivo e fact): def8f3
f1 f1 int divi[MAX];

b1 fb void crivo(int lim) {
3e f5     for (int i = 1; i <= lim; i++) divi[i] = 1;

71 d4     for (int i = 2; i <= lim; i++) if (divi[i] == 1)
41 01         for (int j = i; j <= lim; j += i) divi[j] = i;
8d cb }

b4 47 void fact(vector<int>& v, int n) {
34 ac     if (n != divi[n]) fact(v, n/divi[n]);
f5 ab     v.push_back(divi[n]);
de cb }

// Crivo linear
//
// Mesma coisa que o de cima, mas tambem

```

```

// calcula a lista de primos
//
// O(n)

// 792458
b2 f1 int divi[MAX];
86 fd vector<int> primes;

9f fb void crivo(int lim) {
de d5     divi[1] = 1;
1d f7     for (int i = 2; i <= lim; i++) {
c0 3e         if (divi[i] == 0) divi[i] = i, primes.push_back(i);
75 3b         for (int j : primes) {
99 52             if (j > divi[i] or i*j > lim) break;
eb 00             divi[i*j] = j;
83 cb         }
50 cb     }
57 cb }

// Crivo de divisores
//
// Encontra numero de divisores
// ou soma dos divisores
//
// O(n log(n))

// 9bf7b6
cd f1 int divi[MAX];

0f fb void crivo(int lim) {
b9 f5     for (int i = 1; i <= lim; i++) divi[i] = 1;

9c 42     for (int i = 2; i <= lim; i++)
48 59         for (int j = i; j <= lim; j += i) {
// para numero de divisores
d1 9e             divi[j]++;
// para soma dos divisores
cd 27             divi[j] += i;
4b cb         }
2f cb }

// Crivo de totiente
//
// Encontra o valor da funcao
// totiente de Euler
//
// O(n log(log(n)))

```

```

// 266461
7a 5f int tot[MAX];

19 fb void crivo(int lim) {
c1 a2     for (int i = 1; i <= lim; i++) {
cf bc         tot[i] += i;
aa fe         for (int j = 2*i; j <= lim; j += i)
3a 83             tot[j] -= tot[i];
14 cb     }
9e cb }

// Crivo de funcao de mobius
//
// O(n log(log(n)))

// 58d036
49 4e char meb[MAX];

3d fb void crivo(int lim) {
4a 64     for (int i = 2; i <= lim; i++) meb[i] = 2;
74 ac     meb[1] = 1;
d6 84     for (int i = 2; i <= lim; i++) if (meb[i] == 2)
bd 8d         for (int j = i; j <= lim; j += i) if (meb[j]) {
68 68             if (meb[j] == 2) meb[j] = 1;
59 ae             meb[j] *= j/i%i ? -1 : 0;
01 cb         }
53 cb }

// Crivo linear de funcao multiplicativa
//
// Computa f(i) para todo 1 <= i <= n, sendo f
// uma funcao multiplicativa (se gcd(a,b) = 1,
// entao f(a*b) = f(a)*f(b))
// f_prime tem que computar f de um primo, e
// add_prime tem que computar f(p^(k+1)) dado f(p^k) e p
// Se quiser computar f(p^k) dado p e k, usar os comentarios
//
// O(n)

// 66886a

8c fd vector<int> primes;
67 62 int f[MAX], pot[MAX];
//int expo[MAX];

dc 5c void sieve(int lim) {

```

```

// Funcoes para soma dos divisores:
7e fc     auto f_prime = [](int p) { return p+1; };
0f 31     auto add_prime = [](int fpak, int p) { return fpak*p+1; };
//auto f_pak = [](int p, int k) {};

1d 02     f[1] = 1;
10 f7     for (int i = 2; i <= lim; i++) {
4c e6         if (!pot[i]) {
10 e7             primes.push_back(i);
b4 f0             f[i] = f_prime(i), pot[i] = i;
//expo[i] = 1;

08 cb         }
d7 3b         for (int p : primes) {
78 b9             if (i*p > lim) break;
3d 56             if (i%p == 0) {
17 b9                 f[i*p] = f[i / pot[i]] * add_prime(f[pot[i]], p);
// se for descomentar, tirar a linha de cima tambem
//f[i*p] = f[i / pot[i]] * f_pak(p, expo[i]+1);
//expo[i*p] = expo[i]+1;
f7 51                 pot[i*p] = pot[i] * p;
87 c2                 break;
45 9d             } else {
86 9e                 f[i*p] = f[i] * f[p];
ec 63                 pot[i*p] = p;
//expo[i*p] = 1;

47 cb             }
4c cb         }
85 cb     }
d0 cb }

```

## 4.9 Deteccao de ciclo - Tortoise and Hare

```

// Linear no tanto que tem que andar pra ciclar,
// O(1) de memoria
// Retorna um par com o tanto que tem que andar
// do f0 ate o inicio do ciclo e o tam do ciclo
// 899f20

58 58 pair<ll, ll> find_cycle() {
12 27     ll tort = f(f0);
bc b2     ll hare = f(f(f0));
7f b1     ll t = 0;
f2 68     while (tort != hare) {
66 b4         tort = f(tort);
81 4b         hare = f(f(hare));
62 c8         t++;
76 cb     }

```

```

76 0e    ll st = 0;
38 90    tort = f0;
37 68    while (tort != hare) {
d5 b4        tort = f(tort);
08 1a        hare = f(hare);
95 39        st++;
fc cb    }

6e 73    ll len = 1;
34 3c    hare = f(tort);
d1 68    while (tort != hare) {
78 1a        hare = f(hare);
72 04        len++;
cb cb    }
47 eb    return {st, len};
89 cb }

```

## 4.10 Division Trick

```

// Gera o conjunto n/i, pra todo i, em O(sqrt(n))
// copieie do github do tfg50
// 5bf9bf

```

```

79 79 for(int l = 1, r; l <= n; l = r + 1) {
3f 74     r = n / (n / l);
        // n / i has the same value for l <= i <= r
5b cb }

```

## 4.11 Eliminacao Gaussiana

```

// Resolve sistema linear
// Retornar um par com o numero de solucoes
// e alguma solucao, caso exista
//
// O(n^2 * m)
// 1d10b5

```

```

67 67 template<typename T>
6e 72 pair<int, vector<T>> gauss(vector<vector<T>> a, vector<T> b) {
fe 6c     const double eps = 1e-6;
34 f9     int n = a.size(), m = a[0].size();
fd 2f     for (int i = 0; i < n; i++) a[i].push_back(b[i]);

d9 3c     vector<int> where(m, -1);
fd 23     for (int col = 0, row = 0; col < m and row < n; col++) {
9e f0         int sel = row;
74 b9         for (int i=row; i<n; ++i)

```

```

7f e5         if (abs(a[i][col]) > abs(a[sel][col])) sel = i;
0b 2c         if (abs(a[sel][col]) < eps) continue;
9a 1a         for (int i = col; i <= m; i++)
df dd             swap(a[sel][i], a[row][i]);
93 2c         where[col] = row;

1e 0c         for (int i = 0; i < n; i++) if (i != row) {
54 96             T c = a[i][col] / a[row][col];
96 d5             for (int j = col; j <= m; j++)
29 c8                 a[i][j] -= a[row][j] * c;
8b cb         }
53 b7         row++;
e0 cb     }

f4 b1     vector<T> ans(m, 0);
43 e1     for (int i = 0; i < m; i++) if (where[i] != -1)
d0 12         ans[i] = a[where[i]][m] / a[where[i]][i];
97 60     for (int i = 0; i < n; i++) {
ef 50         T sum = 0;
04 a7         for (int j = 0; j < m; j++)
2c 5a             sum += ans[j] * a[i][j];
36 b1         if (abs(sum - a[i][m]) > eps)
34 6c             return pair(0, vector<T>());
51 cb     }

cd 12     for (int i = 0; i < m; i++) if (where[i] == -1)
3f 01         return pair(INF, ans);
4b 28     return pair(1, ans);
1d cb }

```

## 4.12 Eliminacao Gaussiana Z2

```

// D eh dimensao do espaco vetorial
// add(v) - adiciona o vetor v na base (retorna se ele jah pertencia
//          ao span da base)
// coord(v) - retorna as coordenadas (c) de v na base atual (basis^T.c
//            = v)
// recover(v) - retorna as coordenadas de v nos vetores na ordem em
//              que foram inseridos
// coord(v).first e recover(v).first - se v pertence ao span
//
// Complexidade:
// add, coord, recover: O(D^2 / 64)
// d0a4b3

2a 2a template<int D> struct Gauss_z2 {
88 3c     bitset<D> basis[D], keep[D];

```

```

b0 b1     int rk, in;
41 48     vector<int> id;

34 37     Gauss_z2 () : rk(0), in(-1), id(D, -1) {};

0f 04     bool add(bitset<D> v) {
21 42         in++;
17 fb         bitset<D> k;
7a 65         for (int i = D - 1; i >= 0; i--) if (v[i]) {
14 18             if (basis[i][i]) v ^= basis[i], k ^= keep[i];
d7 4e             else {
27 ea                 k[i] = true, id[i] = in, keep[i] = k;
d9 6c                 basis[i] = v, rk++;
a8 8a                 return true;
e5 cb             }
e8 cb         }
aa d1         return false;
de cb     }

f2 0f     pair<bool, bitset<D>> coord(bitset<D> v) {
05 94         bitset<D> c;
79 65         for (int i = D - 1; i >= 0; i--) if (v[i]) {
c3 a3             if (basis[i][i]) v ^= basis[i], c[i] = true;
fb 8a             else return {false, bitset<D>()};
28 cb         }
97 5d         return {true, c};
59 cb     }

64 33     pair<bool, vector<int>> recover(bitset<D> v) {
e2 22         auto [span, bc] = coord(v);
5f af         if (not span) return {false, {}};
d9 f7         bitset<D> aux;
6e 5a         for (int i = D - 1; i >= 0; i--) if (bc[i]) aux ^= keep[i];
13 ea         vector<int> oc;
0d ef         for (int i = D - 1; i >= 0; i--) if (aux[i])
oc.push_back(id[i]);
b7 00         return {true, oc};
72 cb     }
d0 21 };

```

### 4.13 Equacao Diofantina Linear

```

// Encontra o numero de solucoes de a*x + b*y = c,
// em que x \in [lx, rx] e y \in [ly, ry]
// Usar o comentario para recuperar as solucoes
// (note que o b ao final eh b/gcd(a, b))
// Cuidado com overflow! Tem que caber o quadrado dos valores
//
// O(log(min(a, b)))

```

```

// 2e8259

c5 c5     template<typename T> tuple<ll, T, T> ext_gcd(ll a, ll b) {
48 3b         if (!a) return {b, 0, 1};
05 c4         auto [g, x, y] = ext_gcd<T>(b%a, a);
51 c5         return {g, y - b/a*x, x};
8a cb     }

// numero de solucoes de a*[lx, rx] + b*[ly, ry] = c
80 14     template<typename T = ll> // usar __int128 se for ate 1e18
3d 2a     ll diophantine(ll a, ll b, ll c, ll lx, ll rx, ll ly, ll ry) {
f2 c8         if (lx > rx or ly > ry) return 0;
bd a9         if (a == 0 and b == 0) return c ? 0 : (rx-lx+1)*(ry-ly+1);
58 8c         auto [g, x, y] = ext_gcd<T>(abs(a), abs(b));
10 9c         if (c % g != 0) return 0;
f0 24         if (a == 0) return (rx-lx+1)*(ly <= c/b and c/b <= ry);
26 4c         if (b == 0) return (ry-ly+1)*(lx <= c/a and c/a <= rx);
7f fb         x *= a/abs(a) * c/g, y *= b/abs(b) * c/g, a /= g, b /= g;

3e b2         auto shift = [&](T qt) { x += qt*b, y -= qt*a; };
e4 ef         auto test = [&](T& k, ll mi, ll ma, ll coef, int t) {
70 86             shift((mi - k)*t / coef);
fe 79             if (k < mi) shift(coef > 0 ? t : -t);
8a 74             if (k > ma) return pair<T, T>(rx+2, rx+1);
90 41             T x1 = x;
47 63             shift((ma - k)*t / coef);
f4 c5             if (k > ma) shift(coef > 0 ? -t : t);
de 4a             return pair<T, T>(x1, x);
7b 21         };

95 63         auto [l1, r1] = test(x, lx, rx, b, 1);
e3 38         auto [l2, r2] = test(y, ly, ry, a, -1);
27 c4         if (l2 > r2) swap(l2, r2);
ad 50         T l = max(l1, l2), r = min(r1, r2);
af 33         if (l > r) return 0;
8d 42         ll k = (r-l) / abs(b) + 1;
aa 83         return k; // solucoes: x = l + [0, k)*|b|
2e cb     }

```

### 4.14 Exponenciacao rapida

```

// (x^y mod m) em O(log(y))

// 12b2f8
03 03     ll pow(ll x, ll y, ll m) { // iterativo
08 c8         ll ret = 1;
57 1b         while (y) {

```

```

e9 89      if (y & 1) ret = (ret * x) % m;
29 23      y >= 1;
8f cc      x = (x * x) % m;
7d cb    }
67 ed      return ret;
12 cb    }

// 7d427b
63 03 11 pow(11 x, 11 y, 11 m) { // recursivo
2b 13      if (!y) return 1;
5f 42      11 ans = pow(x*x%m, y/2, m);
60 88      return y%2 ? x*ans%m : ans;
f0 cb    }

```

## 4.15 Fast Walsh Hadamard Transform

```

// FWHT<'|'>(f) eh SOS DP
// FWHT<'&'>(f) eh soma de superset DP
// Se chamar com ^, usar tamanho potencia de 2!!
//
// O(n log(n))
// 50e84f

38 38 template<char op, class T> vector<T> FWHT(vector<T> f, bool inv
    = false) {
54 b7      int n = f.size();
b1 d7      for (int k = 0; (n-1)>>k; k++) for (int i = 0; i < n; i++) if
    (i>>k&1) {
23 29          int j = i^(1<<k);
46 62          if (op == '^') f[j] += f[i], f[i] = f[j] - 2*f[i];
df a3          if (op == '|') f[i] += (inv ? -1 : 1) * f[j];
2d 93          if (op == '&') f[j] += (inv ? -1 : 1) * f[i];
c8 cb      }
94 57      if (op == '^' and inv) for (auto& i : f) i /= n;
0d ab      return f;
50 cb    }

```

## 4.16 FFT

```

// Chamar convolution com vector<complex<double>> para FFT
// Precisa do mint para NTT
//
// O(n log(n))

// Para FFT
// de56b9
48 48 void get_roots(bool f, int n, vector<complex<double>>& roots) {

```

```

8d f2      const static double PI = acosl(-1);
7c 71      for (int i = 0; i < n/2; i++) {
0f b1          double alpha = i*((2*PI)/n);
99 1a          if (f) alpha = -alpha;
ed 06          roots[i] = {cos(alpha), sin(alpha)};
04 cb      }
de cb    }

// Para NTT
// 91cd08
b5 9f template<int p>
52 97 void get_roots(bool f, int n, vector<mod_int<p>>& roots) {
5d 1e      mod_int<p> r;
6e de      int ord;
6a 57      if (p == 998244353) {
c7 9b          r = 102292;
9b 81          ord = (1 << 23);
4f 1c      } else if (p == 754974721) {
dd 43          r = 739831874;
a2 f0          ord = (1 << 24);
b9 b6      } else if (p == 167772161) {
ce a2          r = 243;
cc 03          ord = (1 << 25);
ae 6e      } else assert(false);

15 54      if (f) r = r^(p - 1 -ord/n);
18 ee      else r = r^(ord/n);
65 be      roots[0] = 1;
ee 07      for (int i = 1; i < n/2; i++) roots[i] = roots[i-1]*r;
2b cb    }

// d5c432
95 8a template<typename T> void fft(vector<T> &a, bool f, int N,
    vector<int> &rev) {
f2 bc      for (int i = 0; i < N; i++) if (i < rev[i]) swap(a[i],
    a[rev[i]]);
2d 12      int l, r, m;
8a cb      vector<T> roots(N);
c7 19      for (int n = 2; n <= N; n *= 2) {
67 0f          get_roots(f, n, roots);

c8 5d          for (int pos = 0; pos < N; pos += n) {
15 43              l = pos+0, r = pos+n/2, m = 0;
5b a8              while (m < n/2) {
5f 29                  auto t = roots[m]*a[r];
44 25                  a[r] = a[l] - t;
ec b8                  a[l] = a[l] + t;

```

```

16 92         l++; r++; m++;
77 cb         }
bb cb     }
8d cb }
25 23 if (f) {
ca 1c     auto invN = T(1)/T(N);
a7 55     for (int i = 0; i < N; i++) a[i] = a[i]*invN;
50 cb }
a2 cb }
a8 bf template<typename T> vector<T> convolution(vector<T> &a,
vector<T> &b) {
55 27     vector<T> l(a.begin(), a.end());
eb f4     vector<T> r(b.begin(), b.end());
df 7c     int ln = l.size(), rn = r.size();
6e 28     int N = ln+rn-1;
3d f0     int n = 1, log_n = 0;
8a ac     while (n <= N) { n <= 1; log_n++; }
23 80     vector<int> rev(n);
33 ba     for (int i = 0; i < n; ++i) {
3c 43         rev[i] = 0;
79 92         for (int j = 0; j < log_n; ++j)
f5 83             if (i & (1<<j)) rev[i] |= 1 << (log_n-1-j);
dc cb     }
a9 14     assert(N <= n);
dd fa     l.resize(n);
23 7e     r.resize(n);
4d 56     fft(l, false, n, rev);
96 fc     fft(r, false, n, rev);
f0 91     for (int i = 0; i < n; i++) l[i] *= r[i];
dc 88     fft(l, true, n, rev);
f2 5e     l.resize(N);
55 79     return l;
4f cb }

// NTT
// 3bf256
82 6c template<int p, typename T> vector<mod_int<p>> ntt(vector<T>& a,
vector<T>& b) {
a9 d5     vector<mod_int<p>> A(a.begin(), a.end()), B(b.begin(),
b.end());
04 d2     return convolution(A, B);
24 cb }

// Convolucao de inteiro
//
// Precisa do CRT
//

```

```

// Tabela de valores:
// [0,1] - <int, 1>
// [-1e5, 1e5] - <ll, 2>
// [-1e9, 1e9] - <__int128, 3>
//
// 053a7d
cf b3 template<typename T, int mods>
73 ee vector<T> int_convolution(vector<int>& a, vector<int>& b) {
08 fe     static const int M1 = 998244353, M2 = 754974721, M3 =
167772161;

7b bf     auto c1 = ntt<M1>(a, b);
34 22     auto c2 = (mods >= 2 ? ntt<M2>(a, b) : vector<mod_int<M2>>());
ad f9     auto c3 = (mods >= 3 ? ntt<M3>(a, b) : vector<mod_int<M3>>());

15 2d     vector<T> ans;
3d 5c     for (int i = 0; i < c1.size(); i++) {
29 c0         crt<T> at(c1[i].v, M1);
4b 31         if (mods >= 2) at = at * crt<T>(c2[i].v, M2);
7e 98         if (mods >= 3) at = at * crt<T>(c3[i].v, M3);
67 b2         ans.push_back(at.a);
29 26         if (at.a > at.m/2) ans.back() -= at.m;
a9 cb     }
75 ba     return ans;
d7 cb }

```

## 4.17 Integracao Numerica - Metodo de Simpson 3/8

```

// Integra f no intervalo [a, b], erro cresce proporcional a (b - a)^5
// 352415

67 67 const int N = 3*100; // multiplo de 3
06 28 ld integrate(ld a, ld b, function<ld(ld)> f) {
53 b4     ld s = 0, h = (b - a)/N;
79 06     for (int i = 1 ; i < N; i++) s += f(a + i*h)*(i%3 ? 3 : 2);
16 0d     return (f(a) + s + f(b))*3*h/8;
35 cb }

```

## 4.18 Inverso Modular

```

// Computa o inverso de a modulo b
// Se b eh primo, basta fazer
// a^(b-2)

// cf94fe
f0 f0 ll inv(ll a, ll b) {
cc ae     return a > 1 ? b - inv(b%a, a)*b/a : 1;

```

```

cf cb }

// computa o inverso modular de 1..MAX-1 modulo um primo
// 7e4e3
24 a8 11 inv[MAX]:
1b 0f inv[1] = 1;
e2 0f for (int i = 2; i < MAX; i++) inv[i] = MOD -
    MOD/i*inv[MOD%i]%MOD;

```

## 4.19 Karatsuba

```

// Os pragmas podem ajudar
// Para n ~ 2e5, roda em < 1 s
//
// O(n^1.58)
// 8065d6

//#pragma GCC optimize("Ofast")
//#pragma GCC target ("avx,avx2")
77 77 template<typename T> void kar(T* a, T* b, int n, T* r, T* tmp) {
d5 d4     if (n <= 64) {
59 51         for (int i = 0; i < n; i++) for (int j = 0; j < n; j++)
f7 21             r[i+j] += a[i] * b[j];
3d 50         return;
1c cb     }
4f 19     int mid = n/2;
f1 2d     T *atmp = tmp, *btmp = tmp+mid, *E = tmp+n;
b5 4f     memset(E, 0, sizeof(E[0])*n);
8c c6     for (int i = 0; i < mid; i++) {
62 c7         atmp[i] = a[i] + a[i+mid];
f2 4b         btmp[i] = b[i] + b[i+mid];
74 cb     }
e6 38     kar(atmp, btmp, mid, E, tmp+2*n);
7f b1     kar(a, b, mid, r, tmp+2*n);
cb 22     kar(a+mid, b+mid, mid, r+n, tmp+2*n);
62 c6     for (int i = 0; i < mid; i++) {
a6 73         T temp = r[i+mid];
39 de         r[i+mid] += E[i] - r[i] - r[i+2*mid];
a8 f1         r[i+2*mid] += E[i+mid] - temp - r[i+3*mid];
72 cb     }
28 cb }

da e3 template<typename T> vector<T> karatsuba(vector<T> a, vector<T>
    b) {
bc ba     int n = max(a.size(), b.size());
db a8     while (n&(n-1)) n++;
98 ca     a.resize(n), b.resize(n);

```

```

ca ae     vector<T> ret(2*n), tmp(4*n);
b7 64     kar(&a[0], &b[0], n, &ret[0], &tmp[0]);
d0 ed     return ret;
80 cb }

```

## 4.20 Logaritmo Discreto

```

// Resolve logaritmo discreto com o algoritmo baby step giant step
// Encontra o menor x tal que a^x = b (mod m)
// Se nao tem, retorna -1
//
// O(sqrt(m) * log(sqrt(m)))
// 739fa8
d4 d4
da da int dlog(int b, int a, int m) {
da 9f     if (a == 0) return b ? -1 : 1; // caso nao definido
da d4
11 a6     a %= m, b %= m;
9d a1     int k = 1, shift = 0;
05 31     while (1) {
4f 6e         int g = gcd(a, m);
a4 d4         if (g == 1) break;
a4 d4
68 9b         if (b == k) return shift;
8d 64         if (b % g) return -1;
74 c3         b /= g, m /= g, shift++;
2c 9a         k = (11) k * a / g % m;
e3 cb     }
e3 d4
ab af     int sq = sqrt(m)+1, giant = 1;
73 97     for (int i = 0; i < sq; i++) giant = (11) giant * a % m;
73 d4
80 0b     vector<pair<int, int>> baby;
4a 33     for (int i = 0, cur = b; i <= sq; i++) {
ec 49         baby.emplace_back(cur, i);
a3 16         cur = (11) cur * a % m;
3a cb     }
b3 eb     sort(baby.begin(), baby.end());
b3 d4
f3 9c     for (int j = 1, cur = k; j <= sq; j++) {
7b ac         cur = (11) cur * giant % m;
c3 78         auto it = lower_bound(baby.begin(), baby.end(), pair(cur,
            INF));
21 d2         if (it != baby.begin() and (--it)->first == cur)
c2 ac             return sq * j - it->second + shift;
0f cb     }
0f d4

```



```
4d da    return -1;
73 cb }
```

## 4.21 Miller-Rabin

```
// Testa se n eh primo, n <= 3 * 10^18
//
// O(log(n)), considerando multiplicacao
// e exponenciacao constantes
// 4ebecc

d8 d8 ll mul(ll a, ll b, ll m) {
c7 e7    ll ret = a*b - ll((long double)1/m*a*b+0.5)*m;
3e 07    return ret < 0 ? ret+m : ret;
2f cb }

26 03 ll pow(ll x, ll y, ll m) {
12 13    if (!y) return 1;
c0 db    ll ans = pow(mul(x, x, m), y/2, m);
75 7f    return y%2 ? mul(x, ans, m) : ans;
ca cb }

25 1a bool prime(ll n) {
e3 1a    if (n < 2) return 0;
29 23    if (n <= 3) return 1;
5b 9d    if (n % 2 == 0) return 0;
1a f6    ll r = __builtin_ctzll(n - 1), d = n >> r;

        // com esses primos, o teste funciona garantido para n <= 2^64
        // funciona para n <= 3*10^24 com os primos ate 41
11 77    for (int a : {2, 325, 9375, 28178, 450775, 9780504,
795265022}) {
9f da        ll x = pow(a, d, n);
5c 70        if (x == 1 or x == n - 1 or a % n == 0) continue;

b1 4a        for (int j = 0; j < r - 1; j++) {
c1 10            x = mul(x, x, n);
a0 df            if (x == n - 1) break;
24 cb        }
d3 e1        if (x != n - 1) return 0;
14 cb    }
78 6a    return 1;
4e cb }
```

## 4.22 Pollard's Rho Alg

```
// Usa o algoritmo de deteccao de ciclo de Floyd
```

```
// com uma otimizacao na qual o gcd eh acumulado
// A fatoracao nao sai necessariamente ordenada
// O algoritmo rho encontra um fator de n,
// e funciona muito bem quando n possui um fator pequeno
//
// Complexidades (considerando mul constante):
// rho - esperado O(n^(1/4)) no pior caso
// fact - esperado menos que O(n^(1/4) log(n)) no pior caso
// b00653

d8 d8 ll mul(ll a, ll b, ll m) {
c7 e7    ll ret = a*b - ll((long double)1/m*a*b+0.5)*m;
3e 07    return ret < 0 ? ret+m : ret;
2f cb }

26 03 ll pow(ll x, ll y, ll m) {
12 13    if (!y) return 1;
c0 db    ll ans = pow(mul(x, x, m), y/2, m);
75 7f    return y%2 ? mul(x, ans, m) : ans;
ca cb }

25 1a bool prime(ll n) {
e3 1a    if (n < 2) return 0;
29 23    if (n <= 3) return 1;
5b 9d    if (n % 2 == 0) return 0;

1a f6    ll r = __builtin_ctzll(n - 1), d = n >> r;
11 77    for (int a : {2, 325, 9375, 28178, 450775, 9780504,
795265022}) {
9f da        ll x = pow(a, d, n);
5c 70        if (x == 1 or x == n - 1 or a % n == 0) continue;

b1 4a        for (int j = 0; j < r - 1; j++) {
c1 10            x = mul(x, x, n);
a0 df            if (x == n - 1) break;
24 cb        }
d3 e1        if (x != n - 1) return 0;
14 cb    }
78 6a    return 1;
4e cb }

af 9c ll rho(ll n) {
60 0f    if (n == 1 or prime(n)) return n;
f5 f7    auto f = [n](ll x) {return mul(x, x, n) + 1;};

d8 8a    ll x = 0, y = 0, t = 30, prd = 2, x0 = 1, q;
c6 53    while (t % 40 != 0 or gcd(prd, n) == 1) {
```

```

5c 8a      if (x==y) x = ++x0, y = f(x);
8e e1      q = mul(prd, abs(x-y), n);
58 21      if (q != 0) prd = q;
c8 45      x = f(x), y = f(f(y)), t++;
9e cb    }
9f 00      return gcd(prd, n);
6b cb    }

```

```

ae 5b vector<ll> fact(ll n) {
55 1b      if (n == 1) return {};
6a 0e      if (prime(n)) return {n};
eb 0e      ll d = rho(n);
12 1d      vector<ll> l = fact(d), r = fact(n / d);
9d 3a      l.insert(l.end(), r.begin(), r.end());
39 79      return l;
b0 cb    }

```

## 4.23 Produto de dois long long mod m

```

// 0(1)
// 2f3a79

d8 d8 ll mul(ll a, ll b, ll m) { // a*b % m
c7 e7      ll ret = a*b - ll((long double)1/m*a*b+0.5)*m;
3e 07      return ret < 0 ? ret+m : ret;
2f cb    }

```

## 4.24 Simplex

```

// Maximiza c^T x s.t. Ax <= b, x >= 0
//
// 0(2^n), porem executa em 0(n^3) no caso medio
// 3a08e5

39 39 const double eps = 1e-7;

d7 49 namespace Simplex {
8b 69     vector<vector<double>> T;
75 14     int n, m;
2d 43     vector<int> X, Y;

e9 c5     void pivot(int x, int y) {
07 8e         swap(X[y], Y[x-1]);
ed d0         for (int i = 0; i <= m; i++) if (i != y) T[x][i] /=
            T[x][y];
87 33         T[x][y] = 1/T[x][y];

```

```

62 38         for (int i = 0; i <= n; i++) if (i != x and abs(T[i][y]) >
            eps) {
90 77             for (int j = 0; j <= m; j++) if (j != y) T[i][j] -=
                T[i][y] * T[x][j];
95 3d             T[i][y] = -T[i][y] * T[x][y];
23 cb         }
50 cb     }

            // Retorna o par (valor maximo, vetor solucao)
3b 6f     pair<double, vector<double>> simplex(
8a e9         vector<vector<double>> A, vector<double> b,
            vector<double> c) {
7f 5b         n = b.size(), m = c.size();
64 00         T = vector(n + 1, vector<double>(m + 1));
d0 2d         X = vector<int>(m);
a3 0c         Y = vector<int>(n);
e4 11         for (int i = 0; i < m; i++) X[i] = i;
70 51         for (int i = 0; i < n; i++) Y[i] = i+m;
bf 5b         for (int i = 0; i < m; i++) T[0][i] = -c[i];
b9 60         for (int i = 0; i < n; i++) {
c2 ba             for (int j = 0; j < m; j++) T[i+1][j] = A[i][j];
e1 ec             T[i+1][m] = b[i];
2a cb         }
4e 66         while (true) {
97 71             int x = -1, y = -1;
1d 2d             double mn = -eps;
a9 c2             for (int i = 1; i <= n; i++) if (T[i][m] < mn) mn =
                T[i][m], x = i;
4f af             if (x < 0) break;
b2 88             for (int i = 0; i < m; i++) if (T[x][i] < -eps) { y =
                i; break; }

ec 4a             if (y < 0) return {-1e18, {}}; // sem solucao para Ax
            <= b
aa 7f             pivot(x, y);
a4 cb         }
97 66         while (true) {
88 71             int x = -1, y = -1;
17 2d             double mn = -eps;
fc 56             for (int i = 0; i < m; i++) if (T[0][i] < mn) mn =
                T[0][i], y = i;
32 9b             if (y < 0) break;
aa 03             mn = 1e200;
e4 5a             for (int i = 1; i <= n; i++) if (T[i][y] > eps and
                T[i][m] / T[i][y] < mn)
60 48                 mn = T[i][m] / T[i][y], x = i;

```

```

2d 53         if (x < 0) return {1e18, {}}; // c^T x eh ilimitado
d2 7f         pivot(x, y);
2f cb         }
39 29         vector<double> r(m);
ab 32         for(int i = 0; i < n; i++) if (Y[i] < m) r[Y[i]] =
            T[i+1][m];
a7 e5         return {T[0][m], r};
9d cb     }
3a cb }

```

## 4.25 Teorema Chines do Resto

```

// Combina equacoes modulares lineares: x = a (mod m)
// 0 m final eh o lcm dos m's, e a resposta eh unica mod o lcm
// Os m nao precisam ser coprimos
// Se nao tiver solucao, o 'a' vai ser -1
// 7cd7b3

```

```

15 15 template<typename T> tuple<T, T, T> ext_gcd(T a, T b) {
3c 3b     if (!a) return {b, 0, 1};
ab 55     auto [g, x, y] = ext_gcd(b%a, a);
04 c5     return {g, y - b/a*x, x};
53 cb }

```

```

da bf template<typename T = ll> struct crt {
0f 62     T a, m;

```

```

97 5f     crt() : a(0), m(1) {}
2d 7e     crt(T a_, T m_) : a(a_), m(m_) {}
12 91     crt operator * (crt C) {
83 23         auto [g, x, y] = ext_gcd(m, C.m);
89 dc         if ((a - C.a) % g) a = -1;
a3 4f         if (a == -1 or C.a == -1) return crt(-1, 0);
59 d0         T lcm = m/g*C.m;
8c eb         T ans = a + (x*(C.a-a)/g % (C.m/g))*m;
84 d8         return crt((ans % lcm + lcm) % lcm, lcm);
3b cb     }
7c 21 };

```

## 4.26 Totiente

```

// 0(sqrt(n))
// faeca3

```

```

a7 a7 int tot(int n){
d6 0f     int ret = n;

```

```

28 50     for (int i = 2; i*i <= n; i++) if (n % i == 0) {
ed b0         while (n % i == 0) n /= i;
6a 12         ret -= ret / i;
04 cb     }
0f af     if (n > 1) ret -= ret / n;

e2 ed     return ret;
fa cb }

```

## 5 Primitivas

### 5.1 Aritmetica Modular

```

// 0 mod tem q ser primo
// 5a6efb

```

```

42 42 template<int p> struct mod_int {
82 02     ll pow(ll b, ll e) {
e2 a6         if (e == 0) return 1;
c3 63         ll r = pow(b*b%p, e/2);
a3 47         if (e%2 == 1) r = (r*b)%p;
2c 4c         return r;
55 cb     }
d9 ae     ll inv(ll b) { return pow(b, p-2); }

```

```

ac 4d     using m = mod_int;
a7 d9     int v;
a3 fe     mod_int() : v(0) {}
28 e1     mod_int(ll v_) {
b1 01         if (v_ >= p or v_ <= -p) v_ %= p;
14 bc         if (v_ < 0) v_ += p;
e2 2e         v = v_;
bc cb     }
35 74     m& operator+=(const m &a) {
a3 2f         v += a.v;
89 ba         if (v >= p) v -= p;
db 35         return *this;
9b cb     }
59 ef     m& operator-=(const m &a) {
47 8b         v -= a.v;
4b cc         if (v < 0) v += p;
b9 35         return *this;
5d cb     }
7d 4c     m& operator*=(const m &a) {
e9 8a         v = v * ll(a.v) % p;
9d 35         return *this;

```

```

db cb    }
84 3f    m& operator/=(const m &a) {
d4 5d        v = v* inv(a.v) % p;
f3 35        return *this;
8c cb    }
d9 d6    m operator-(){ return m(-v); }
e2 b3    m& operator^=(ll e) {
8e 06        if (e < 0){
6a 6e            v = inv(v);
e8 00            e = -e;
15 cb        }
d4 eb        v = pow(v, e%(p-1));
6f 35        return *this;
09 cb    }
26 42    bool operator==(const m &a) { return v == a.v; }
85 69    bool operator!=(const m &a) { return v != a.v; }

1e 1c    friend istream &operator>>(istream &in, m& a) {
9f d1        ll val; in >> val;
84 d4        a = m(val);
66 09        return in;
ff cb    }
2a 44    friend ostream &operator<<(ostream &out, m a) {
03 5a        return out << a.v;
74 cb    }
a6 39    friend m operator+(m a, m b) { return a+=b; }
07 f9    friend m operator-(m a, m b) { return a-=b; }
3f 9c    friend m operator*(m a, m b) { return a*=b; }
77 51    friend m operator/(m a, m b) { return a/=b; }
63 08    friend m operator^(m a, ll e) { return a^=e; }
f9 21 };

5a 05    typedef mod_int<(int)1e9+7> mint;

```

## 5.2 Big Integer

```

// Complexidades: (para n digitos)
// Soma, subtracao, comparacao - O(n)
// Multiplicacao - O(n log(n))
// Divisao, resto - O(n^2)
// 6c3c3a

```

```

86 86    struct bint {
7e 66        static const int BASE = 1e9;
4b 99        vector<int> v;
b3 3b        bool neg;

```

```

01 60        bint() : neg(0) {}
f0 d5        bint(int val) : bint() { *this = val; }
55 e8        bint(long long val) : bint() { *this = val; }

05 a0        void trim() {
63 f4            while (v.size() and v.back() == 0) v.pop_back();
8d df            if (!v.size()) neg = 0;
ab cb        }

// converter de/para string | cin/cout
39 29        bint(const char* s) : bint() { from_string(string(s)); }
52 54        bint(const string& s) : bint() { from_string(s); }
75 4a        void from_string(const string& s) {
1b 0a            v.clear(), neg = 0;
0b d7            int ini = 0;
38 8e            while (ini < s.size() and (s[ini] == '-' or s[ini] == '+',
or s[ini] == '0'))
79 71                if (s[ini++] == '-') neg = 1;
b3 88            for (int i = s.size()-1; i >= ini; i -= 9) {
e5 05                int at = 0;
9a 5b                for (int j = max(ini, i - 8); j <= i; j++) at = 10*at
+ (s[j] - '0');
57 1f                v.push_back(at);
25 cb            }
35 df            if (!v.size()) neg = 0;
43 cb        }
4b 2f        string to_string() const {
03 8b            if (!v.size()) return "0";
07 79            string ret;
9d 73            if (neg) ret += '-';
6a 3e            for (int i = v.size()-1; i >= 0; i--) {
6c 58                string at = ::to_string(v[i]);
ac ce                int add = 9 - at.size();
18 75                if (i+1 < v.size()) for (int j = 0; j < add; j++) ret
+= '0';
6a f9                ret += at;
96 cb            }
8e ed            return ret;
1d cb        }
71 d2        friend istream& operator>>(istream& in, bint& val) {
be eb            string s; in >> s;
29 96            val = s;
44 09            return in;
a9 cb        }
57 99        friend ostream& operator<<(ostream& out, const bint& val) {
d9 8b            string s = val.to_string();
c2 39            out << s;

```

```

ae fe      return out;
d8 cb      }

        // operators
28 60      friend bint abs(bint val) {
86 c5          val.neg = 0;
f9 d9          return val;
a7 cb      }
b6 be      friend bint operator-(bint val) {
7f 81          if (val != 0) val.neg ^= 1;
e9 d9          return val;
e3 cb      }
f0 41      bint& operator=(const bint& val) { v = val.v, neg = val.neg;
        return *this; }
3b 24      bint& operator=(long long val) {
4e 0a          v.clear(), neg = 0;
d2 3a          if (val < 0) neg = 1, val *= -1;
5a fd          for (; val; val /= BASE) v.push_back(val % BASE);
fb 35          return *this;
fc cb      }
7f 3b      int cmp(const bint& r) const { // menor: -1 | igual: 0 |
        maior: 1
6d b1          if (neg != r.neg) return neg ? -1 : 1;
cb 0b          if (v.size() != r.v.size()) {
89 ff              int ret = v.size() < r.v.size() ? -1 : 1;
92 91              return neg ? -ret : ret;
e8 cb          }
2e 47          for (int i = int(v.size())-1; i >= 0; i--) {
50 40              if (v[i] != r.v[i]) {
05 2e                  int ret = v[i] < r.v[i] ? -1 : 1;
c8 91                  return neg ? -ret : ret;
d3 cb              }
2f cb          }
fe bb          return 0;
87 cb      }
77 15      friend bool operator<(const bint& l, const bint& r) { return
        l.cmp(r) == -1; }
fc c7      friend bool operator>(const bint& l, const bint& r) { return
        l.cmp(r) == 1; }
63 ed      friend bool operator<=(const bint& l, const bint& r) { return
        l.cmp(r) <= 0; }
47 95      friend bool operator>=(const bint& l, const bint& r) { return
        l.cmp(r) >= 0; }
45 a6      friend bool operator==(const bint& l, const bint& r) { return
        l.cmp(r) == 0; }
76 10      friend bool operator!=(const bint& l, const bint& r) { return
        l.cmp(r) != 0; }

```

```

82 38      bint& operator +=(const bint& r) {
1f 6b          if (!r.v.size()) return *this;
7c a9          if (neg != r.neg) return *this -= -r;
27 25          for (int i = 0, c = 0; i < r.v.size() or c; i++) {
fa e2              if (i == v.size()) v.push_back(0);
2b 08              v[i] += c + (i < r.v.size() ? r.v[i] : 0);
66 ba              if ((c = v[i] >= BASE)) v[i] -= BASE;
7c cb          }
56 35          return *this;
3e cb      }
de 54      friend bint operator+(bint a, const bint& b) { return a += b; }
a9 9c      bint& operator -=(const bint& r) {
5e 6b          if (!r.v.size()) return *this;
ea 52          if (neg != r.neg) return *this += -r;
b2 35          if ((!neg and *this < r) or (neg and r < *this)) {
eb b1              *this = r - *this;
3a a1              neg ^= 1;
bc 35              return *this;
18 cb          }
09 25          for (int i = 0, c = 0; i < r.v.size() or c; i++) {
ee 9e              v[i] -= c + (i < r.v.size() ? r.v[i] : 0);
f1 c8              if ((c = v[i] < 0)) v[i] += BASE;
a1 cb          }
4f 0e          trim();
be 35          return *this;
95 cb      }
12 f4      friend bint operator-(bint a, const bint& b) { return a -= b; }

        // operators de * / %
25 6b      bint& operator *=(int val) {
5c bc          if (val < 0) val *= -1, neg ^= 1;
b7 56          for (int i = 0, c = 0; i < v.size() or c; i++) {
0a e2              if (i == v.size()) v.push_back(0);
9f 35              long long at = (long long) v[i] * val + c;
90 6a              v[i] = at % BASE;
d3 b3              c = at / BASE;
5b cb          }
d2 0e          trim();
f8 35          return *this;
0b cb      }
c6 48      friend bint operator *(bint a, int b) { return a *= b; }
7f d5      friend bint operator *(int a, bint b) { return b *= a; }
c5 13      using cplx = complex<double>;
f7 bf      void fft(vector<cplx>& a, bool f, int N, vector<int>& rev)
        const {
5e bc          for (int i = 0; i < N; i++) if (i < rev[i]) swap(a[i],

```

```

a[rev[i]]);
4f ba      vector<cplx> roots(N);
9a 19      for (int n = 2; n <= N; n *= 2) {
ff 4e          const static double PI = acos(-1);
8e 71          for (int i = 0; i < n/2; i++) {
f9 40              double alpha = (2*PI*i)/n;
70 1a              if (f) alpha = -alpha;
d4 3f              roots[i] = cplx(cos(alpha), sin(alpha));
6d cb          }
af 3e          for (int pos = 0; pos < N; pos += n)
7a 89              for (int l = pos, r = pos+n/2, m = 0; m < n/2;
l++, r++, m++) {
7b 29                  auto t = roots[m]*a[r];
bb 25                  a[r] = a[l] - t;
ce b8                  a[l] = a[l] + t;
30 cb              }
2b cb          }
c7 3f          if (!f) return;
4b 08          auto invN = cplx(1)/cplx(N);
31 87          for (int i = 0; i < N; i++) a[i] *= invN;
12 cb      }
88 0e      vector<long long> convolution(const vector<int>& a, const
vector<int>& b) const {
06 ff          vector<cplx> l(a.begin(), a.end()), r(b.begin(), b.end());
dd 99          int ln = l.size(), rn = r.size(), N = ln+rn+1, n = 1,
log_n = 0;
ec 82          while (n <= N) n <= 1, log_n++;
9d 80          vector<int> rev(n);
39 60          for (int i = 0; i < n; i++) {
08 43              rev[i] = 0;
d2 f4              for (int j = 0; j < log_n; j++) if (i>>j&1)
e3 4f                  rev[i] |= 1 << (log_n-1-j);
8f cb          }
b6 23          l.resize(n), r.resize(n);
54 a8          fft(l, false, n, rev), fft(r, false, n, rev);
88 91          for (int i = 0; i < n; i++) l[i] *= r[i];
40 88          fft(l, true, n, rev);
61 7a          vector<long long> ret;
dd c1          for (auto& i : l) ret.push_back(round(i.real()));
0b ed          return ret;
6a cb      }
e9 63      vector<int> convert_base(const vector<int>& a, int from, int
to) const {
0a 49          static vector<long long> pot(10, 1);
08 67          if (pot[1] == 1) for (int i = 1; i < 10; i++) pot[i] =
10*pot[i-1];
6f 4b          vector<int> ret;

```

```

2f 15          long long at = 0;
f8 60          int digits = 0;
64 94          for (int i : a) {
a6 41              at += i * pot[digits];
39 03              digits += from;
5d 68              while (digits >= to) {
43 0c                  ret.push_back(at % pot[to]);
02 cf                  at /= pot[to];
a5 fd                  digits -= to;
8c cb              }
6c cb          }
8a 94          ret.push_back(at);
8e 38          while (ret.size() and ret.back() == 0) ret.pop_back();
1e ed          return ret;
10 cb      }
57 ed      bint operator*(const bint& r) const { // O(n log(n))
19 2a          bint ret;
46 96          ret.neg = neg ^ r.neg;
3e d5          auto conv = convolution(convert_base(v, 9, 4),
convert_base(r.v, 9, 4));
46 a0          long long c = 0;
5e a7          for (auto i : conv) {
eb f6              long long at = i+c;
8a 4c              ret.v.push_back(at % 10000);
a4 a2              c = at / 10000;
60 cb          }
9a 3c          for (; c; c /= 10000) ret.v.push_back(c%10000);
c6 0e          ret.v = convert_base(ret.v, 4, 9);
cf 25          if (!ret.v.size()) ret.neg = 0;
d1 ed          return ret;
23 cb      }
2f 35      bint& operator*=(const bint& r) { return *this = *this * r; };
7f 9a      bint& operator/=(int val) {
b6 d9          if (val < 0) neg ^= 1, val *= -1;
a9 f1          for (int i = int(v.size())-1, c = 0; i >= 0; i--) {
85 2a              long long at = v[i] + c * (long long) BASE;
2b e0              v[i] = at / val;
72 fb              c = at % val;
4f cb          }
95 0e          trim();
da 35          return *this;
a8 cb      }
ba e7      friend bint operator/(bint a, int b) { return a /= b; }
3b 4a      int operator %=(int val) {
6e 23          if (val < 0) val *= -1;
c7 15          long long at = 0;
92 f3          for (int i = int(v.size())-1; i >= 0; i--)

```

```

e1 1b          at = (BASE * at + v[i]) % val;
7b d2          if (neg) at *= -1;
9e ce          return at;
0c cb      }
77 2f  friend int operator%(bint a, int b) { return a %= b; }
49 13  friend pair<bint, bint> divmod(const bint& a_, const bint& b_)
    { // O(n^2)
c7 61          if (a_ == 0) return {0, 0};
4f d8          int norm = BASE / (b_.v.back() + 1);
1e b4          bint a = abs(a_) * norm;
da 02          bint b = abs(b_) * norm;
d7 14          bint q, r;
de c9          for (int i = a.v.size() - 1; i >= 0; i--) {
0e b7              r *= BASE, r += a.v[i];
fb 4f              long long upper = b.v.size() < r.v.size() ?
                  r.v[b.v.size()] : 0;
8a 86              int lower = b.v.size() - 1 < r.v.size() ?
                  r.v[b.v.size() - 1] : 0;
82 43              int d = (upper * BASE + lower) / b.v.back();
a4 5d              r -= b*d;
62 30              while (r < 0) r += b, d--; // roda O(1) vezes
3e 73              q.v.push_back(d);
56 cb          }
85 a4          reverse(q.v.begin(), q.v.end());
41 ae          q.neg = a_.neg ^ b_.neg;
35 88          r.neg = a_.neg;
2c 8e          q.trim(), r.trim();
2f 0e          return {q, r / norm};
ec cb      }
26 1d  bint operator/(const bint& val) { return divmod(*this,
    val).first; }
bf 7f  bint& operator/=(const bint& val) { return *this = *this /
    val; }
db 1f  bint operator%(const bint& val) { return divmod(*this,
    val).second; }
3e df  bint& operator%=(const bint& val) { return *this = *this %
    val; }
6c 21 };

```

## 5.3 Matroid

```

// Matroids de Grafo e Particao
// De modo geral, toda Matroid contem um build() linear
// e uma funcao constante oracle()
// oracle(i) responde se o conjunto continua independente
// apos adicao do elemento i
// oracle(i, j) responde se o conjunto continua indepenete

```

```

// apos trocar o elemento i pelo elemento j
//
// Intersecao sem peso O(r^2 n)
// em que n eh o tamanho do conjunto e r eh o tamanho da resposta

// Matroid Grafica
// Matroid das florestas de um grafo
// Um conjunto de arestas eh independente se formam uma floresta
//
// build() : O(n)
// oracle() : O(1)
// 691847

fd fd struct graphic_matroid {
9d 5d     int n, m, t;
9d 32     vector<array<int, 2>> edges;
0b 78     vector<vector<int>> g;
d4 62     vector<int> comp, in, out;
81 51     graphic_matroid(int n_, vector<array<int, 2>> edges_)
55 a1         : n(n_), m(edges_.size()), edges(edges_), g(n), comp(n),
            in(n), out(n) {}
48 31     void dfs(int u) {
0e ab         in[u] = t++;
45 17         for (auto v : g[u]) if (in[v] == -1)
b4 86             comp[v] = comp[u], dfs(v);
4a 67         out[u] = t;
ab cb     }
5b 94     void build(vector<int> I) {
54 a3         t = 0;
16 74         for (int u = 0; u < n; u++) g[u].clear(), in[u] = -1;
e9 66         for (int e : I) {
08 d0             auto [u, v] = edges[e];
03 12             g[u].push_back(v), g[v].push_back(u);
f7 cb         }
5b 80         for (int u = 0; u < n; u++) if (in[u] == -1)
e3 a7             comp[u] = u, dfs(u);
ad cb     }
ea f3     bool is_ancestor(int u, int v) {
69 a6         return in[u] <= in[v] and in[v] < out[u];
ff cb     }
f4 e6     bool oracle(int e) {
b5 45         return comp[edges[e][0]] != comp[edges[e][1]];
8c cb     }
b7 f7     bool oracle(int e, int f) {
a1 57         if (oracle(f)) return true;
da 62         int u = edges[e][in[edges[e][0]] < in[edges[e][1]]];
11 ff         return is_ancestor(u, edges[f][0]) != is_ancestor(u,

```

```

    edges[f][1]);
b0 cb  }
69 21 };

// Matroid de particao ou cores
// Um conjunto eh independente se a quantidade de elementos
// de cada cor nao excede a capacidade da cor
// Quando todas as capacidades sao 1, um conjunto eh independente
// se todas as suas cores sao distintas
//
// build() : O(n)
// oracle() : O(1)
// caa72a

f1 99 struct partition_matroid {
f2 50     vector<int> cap, color, d;
27 60     partition_matroid(vector<int> cap_, vector<int> color_)
f0 04         : cap(cap_), color(color_), d(cap.size()) {}
c7 94     void build(vector<int> I) {
a9 de         fill(d.begin(), d.end(), 0);
01 e9         for (int u : I) d[color[u]]++;
e1 cb     }
c1 51     bool oracle(int u) {
f8 0a         return d[color[u]] < cap[color[u]];
b8 cb     }
a4 f7     bool oracle(int u, int v) {
44 2f         return color[u] == color[v] or oracle(v);
9d cb     }
dd 21 };

// Intersecao de matroid sem pesos
// Dadas duas matroids M1 e M2 definidas sobre o mesmo
// conjunto I, retorna o maior subconjunto de I
// que eh independente tanto para M1 quanto para M2
//
// O(r^2*n)
// 899f94

// Matroid "pesada" deve ser a M2
1d 13 template<typename Matroid1, typename Matroid2>
a3 80 vector<int> matroid_intersection(int n, Matroid1 M1, Matroid2
    M2) {
36 f5     vector<bool> b(n);
53 a6     vector<int> I[2];
dc a8     bool converged = false;
68 0c     while (!converged) {
d9 74         I[0].clear(), I[1].clear();

```

```

db 99         for (int u = 0; u < n; u++) I[b[u]].push_back(u);

8a 09     M1.build(I[1]), M2.build(I[1]);
ab 28     vector<bool> target(n), pushed(n);
94 26     queue<int> q;
20 5c     for (int u : I[0]) {
6c 2b         target[u] = M2.oracle(u);
69 c1         if (M1.oracle(u)) pushed[u] = true, q.push(u);
db cb     }
fc 3f     vector<int> p(n, -1);
43 07     converged = true;
ba 40     while (q.size()) {
3f be         int u = q.front(); q.pop();
45 5c         if (target[u]) {
7d 10             converged = false;
0b c3             for (int v = u; v != -1; v = p[v]) b[v] = !b[v];
56 c2             break;
61 cb         }
86 e7         for (int v : I[!b[u]]) if (!pushed[v]) {
46 34             if ((b[u] and M1.oracle(u, v)) or (b[v] and
                M2.oracle(v, u)))
34 ba                 p[v] = u, pushed[v] = true, q.push(v);
d4 cb         }
45 cb     }
59 cb     }
7e b6     return I[1];
6e cb }

// Intersecao de matroid com pesos
// Dadas duas matroids M1 e M2 e uma funcao de pesos w, todas
// definidas sobre
// um conjunto I retorna o maior subconjunto de I (desempatado pelo
// menor peso)
// que eh independente tanto para M1 quanto para M2
// A resposta eh construida incrementando o tamanho conjunto I de 1 em
// 1
// Se nao tiver custo negativo, nao precisa de SPFA
//
// O(r^3*n) com SPFA
// O(r^2*n*log(n)) com Dijkstra e potencial
// 3a09d1

0e 42 template<typename T, typename Matroid1, typename Matroid2>
99 2b vector<int> weighted_matroid_intersection(int n, vector<T> w,
    Matroid1 M1, Matroid2 M2) {
4f 6c     vector<bool> b(n), target(n), is_inside(n);
e6 56     vector<int> I[2], from(n);

```



```

e7 e3    vector<pair<T, int>> d(n);
ce 16    auto check_edge = [&](int u, int v) {
60 24        return (b[u] and M1.oracle(u, v)) or (b[v] and
M2.oracle(v, u));
85 21    };
23 66    while (true) {
a4 74        I[0].clear(), I[1].clear();
1d 99        for (int u = 0; u < n; u++) I[b[u]].push_back(u);
// I[1] contem o conjunto de tamanho I[1].size() de menor
// peso
7a 09        M1.build(I[1]), M2.build(I[1]);
24 68        for (int u = 0; u < n; u++) {
0a ea            target[u] = false, is_inside[u] = false, from[u] = -1;
c3 96            d[u] = {numeric_limits<T>::max(), INF};
be cb        }
1c 8d        deque<T> q;
b2 47        sort(I[0].begin(), I[0].end(), [&](int i, int j){ return
w[i] < w[j]; });
03 5c        for (int u : I[0]) {
6b 2b            target[u] = M2.oracle(u);
5f 5a            if (M1.oracle(u)) {
af 4e                if (is_inside[u]) continue;
db 7c                d[u] = {w[u], 0};
9a 42                if (!q.empty() and d[u] > d[q.front()])
q.push_back(u);
e5 65                else q.push_front(u);
a0 4a                is_inside[u] = true;
c1 cb            }
05 cb        }
b5 40        while (q.size()) {
5a 97            int u = q.front(); q.pop_front();
14 6f            is_inside[u] = false;
f9 57            for (int v : I[!b[u]]) if (check_edge(u, v)) {
85 9d                pair<T, int> nd(d[u].first + w[v], d[u].second +
1);
d6 61                if (nd < d[v]) {
8b 6a                    from[v] = u, d[v] = nd;
ec bd                    if (is_inside[v]) continue;
32 ee                    if (q.size() and d[v] > d[q.front()])
q.push_back(v);
e9 27                    else q.push_front(v);
b4 58                    is_inside[v] = true;
c5 cb                }
6a cb            }
31 cb        }
35 cc        pair<T, int> mini = pair(numeric_limits<T>::max(), INF);
6b 48        int targ = -1;

```

```

73 25        for (int u : I[0]) if (target[u] and d[u] < mini)
eb 2b            mini = d[u], targ = u;
51 e1        if (targ != -1) for (int u = targ; u != -1; u = from[u])
05 d8            b[u] = !b[u], w[u] *= -1;
4a f9        else break;
a3 cb    }
dc b6    return I[1];
7b cb }

```

## 5.4 Primitivas de fracao

```

// Funciona com o Big Int
// cdb445

```

```

a4 a4    template<typename T = int> struct frac {
50 a4        T num, den;
4b e3        template<class U, class V>
81 61        frac(U num_ = 0, V den_ = 1) : num(num_), den(den_) {
90 ba            assert(den != 0);
a1 58            if (den < 0) num *= -1, den *= -1;
52 a5            T g = gcd(abs(num), den);
63 57            num /= g, den /= g;
bd cb    }

37 51        friend bool operator< (const frac& l, const frac& r) {
dd fa            return l.num * r.den < r.num * l.den;
57 cb    }
91 4b        friend frac operator+ (const frac& l, const frac& r) {
7a b6            return {l.num*r.den + l.den*r.num, l.den*r.den};
83 cb    }
2b 74        friend frac operator- (const frac& l, const frac& r) {
e7 2c            return {l.num*r.den - l.den*r.num, l.den*r.den};
c4 cb    }
e7 c8        friend frac operator* (const frac& l, const frac& r) {
b4 51            return {l.num*r.num, l.den*r.den};
47 cb    }
74 a1        friend frac operator/ (const frac& l, const frac& r) {
3a 8f            return {l.num*r.den, l.den*r.num};
e7 cb    }
6f 01        friend ostream& operator<< (ostream& out, frac f) {
97 37            out << f.num << '/' << f.den;
50 fe            return out;
80 cb    }
cd 21 };

```

## 5.5 Primitivas de matriz - exponenciacao

```
// d05c24
```

```
94 94 #define MODULAR false
91 5e template<typename T> struct matrix : vector<vector<T>> {
ab 14     int n, m;

57 30     void print() {
a8 60         for (int i = 0; i < n; i++) {
33 70             for (int j = 0; j < m; j++) cout << (*this)[i][j] << "
";
36 1f             cout << endl;
18 cb         }
4c cb     }

28 aa     matrix(int n_, int m_, bool ident = false) :
02 b1         vector<vector<T>>(n_, vector<T>(m_, 0)), n(n_), m(m_) {
97 94         if (ident) {
b0 df             assert(n == m);
bf a8             for (int i = 0; i < n; i++) (*this)[i][i] = 1;
d3 cb         }
5d cb     }
b7 b8     matrix(const vector<vector<T>>& c) : vector<vector<T>>(c),
ab a3         n(c.size()), m(c[0].size()) {}
6a ef     matrix(const initializer_list<initializer_list<T>>& c) {
3a f7         vector<vector<T>> val;
76 21         for (auto& i : c) val.push_back(i);
2e 30         *this = matrix(val);
0c cb     }

11 38     matrix<T> operator*(matrix<T>& r) {
81 1e         assert(m == r.n);
2a 82         matrix<T> M(n, r.m);
5f d6         for (int i = 0; i < n; i++) for (int k = 0; k < m; k++)
b5 df             for (int j = 0; j < r.m; j++) {
9a e3                 T add = (*this)[i][k] * r[k][j];
a3 f9 #if MODULAR
a3 d4 #warning Usar matrix<ll> e soh colocar valores em [0, MOD) na
        matriz!
6c 8b                 M[i][j] += add%MOD;
98 98                 if (M[i][j] >= MOD) M[i][j] -= MOD;
98 8c #else
47 7b                 M[i][j] += add;
c4 f2 #endif
51 cb             }
fd 47         return M;
56 cb     }
a5 52     matrix<T> operator^(ll e){
```

```
ba f1         matrix<T> M(n, n, true), at = *this;
73 c8         while (e) {
f3 2e             if (e&1) M = M*at;
87 cc             e >>= 1;
3c c8             at = at*at;
04 cb         }
1f 47         return M;
e5 cb     }
e8 58     void apply_transform(matrix M, ll e){
37 1c         auto& v = *this;
ae c8         while (e) {
db 9b             if (e&1) v = M*v;
dd cc             e >>= 1;
3b 41             M = M*M;
ca cb         }
29 cb     }
d0 21 };
```

## 5.6 Primitivas Geometricas

```
c8 c8 typedef double ld;
a4 e3 const ld DINF = 1e18;
4a 43 const ld pi = acos(-1.0);
40 10 const ld eps = 1e-9;
```

```
53 b3 #define sq(x) ((x)*(x))
```

```
6f d9 bool eq(ld a, ld b) {
45 ba     return abs(a - b) <= eps;
4d cb }
```

```
// a8b7d6
```

```
67 b2 struct pt { // ponto
1b c1     ld x, y;
21 3d     pt(ld x_ = 0, ld y_ = 0) : x(x_), y(y_) {}
68 5b     bool operator < (const pt p) const {
c8 05         if (!eq(x, p.x)) return x < p.x;
eb f9         if (!eq(y, p.y)) return y < p.y;
a4 bb         return 0;
2c cb     }
0d a8     bool operator == (const pt p) const {
32 ed         return eq(x, p.x) and eq(y, p.y);
ac cb     }
8c cb     pt operator + (const pt p) const { return pt(x+p.x, y+p.y); }
fb a2     pt operator - (const pt p) const { return pt(x-p.x, y-p.y); }
b1 4a     pt operator * (const ld c) const { return pt(x*c, y*c); }
12 a6     pt operator / (const ld c) const { return pt(x/c, y/c); }
```

```

fe 3b    ld operator * (const pt p) const { return x*p.x + y*p.y; }
f7 6d    ld operator ^ (const pt p) const { return x*p.y - y*p.x; }
28 5e    friend istream& operator >> (istream& in, pt& p) {
b8 e3        return in >> p.x >> p.y;
94 cb    }
25 21    };

// 7ab617
2c b3    struct line { // reta
47 73        pt p, q;
b9 0d        line() {}
46 4b        line(pt p_, pt q_) : p(p_), q(q_) {}
da 8d        friend istream& operator >> (istream& in, line& r) {
18 4c            return in >> r.p >> r.q;
41 cb        }
f1 21    };

// PONTO & VETOR

// c684fb
cc 36    ld dist(pt p, pt q) { // distancia
9d 5f        return hypot(p.y - q.y, p.x - q.x);
9a cb    }

// 80f2b6
ea 9d    ld dist2(pt p, pt q) { // quadrado da distancia
b5 f2        return sq(p.x - q.x) + sq(p.y - q.y);
16 cb    }

// cf7f33
7d 48    ld norm(pt v) { // norma do vetor
fb 49        return dist(pt(0, 0), v);
f5 cb    }

// 404df7
e0 58    ld angle(pt v) { // angulo do vetor com o eixo x
7e 58        ld ang = atan2(v.y, v.x);
2b 6f        if (ang < 0) ang += 2*pi;
39 19        return ang;
2a cb    }

// 1b1d4a
98 29    ld sarea(pt p, pt q, pt r) { // area com sinal
ac 60        return ((q-p)^(r-q))/2;
52 cb    }

// 98c42f

```

```

99 e3    bool col(pt p, pt q, pt r) { // se p, q e r sao colin.
51 e7        return eq(sarea(p, q, r), 0);
e3 cb    }

// 85d09d
7d 0c    bool ccw(pt p, pt q, pt r) { // se p, q, r sao ccw
e8 fa        return sarea(p, q, r) > eps;
55 cb    }

// 41a7b4
4f 1e    pt rotate(pt p, ld th) { // rotaciona o ponto th radianos
20 e5        return pt(p.x * cos(th) - p.y * sin(th),
89 ff            p.x * sin(th) + p.y * cos(th));
20 cb    }

// e4ad5e
2b ab    pt rotate90(pt p) { // rotaciona 90 graus
71 a0        return pt(-p.y, p.x);
d5 cb    }

// RETA

// 0fb984
0d ed    bool isvert(line r) { // se r eh vertical
28 87        return eq(r.p.x, r.q.x);
2a cb    }

// 726d68
59 09    bool isinseg(pt p, line r) { // se p pertence ao seg de r
8d f6        pt a = r.p - p, b = r.q - p;
fd b0        return eq((a ^ b), 0) and (a * b) < eps;
ca cb    }

// a0a30b
9d 98    ld get_t(pt v, line r) { // retorna t tal que t*v pertence a
        reta r
80 6e        return (r.p^r.q) / ((r.p-r.q)^v);
a5 cb    }

// 2329fe
4f 25    pt proj(pt p, line r) { // projecao do ponto p na reta r
2c be        if (r.p == r.q) return r.p;
6e 97        r.q = r.q - r.p; p = p - r.p;
52 9f        pt proj = r.q * ((p*r.q) / (r.q*r.q));
0c 2c        return proj + r.p;
10 cb    }

```

```

// 111fd2
44 d5 pt inter(line r, line s) { // r inter s
f5 14   if (eq((r.p - r.q) ^ (s.p - s.q), 0)) return pt(DINF, DINF);
a3 20   r.q = r.q - r.p, s.p = s.p - r.p, s.q = s.q - r.p;
f6 54   return r.q * get_t(r.q, s) + r.p;
0a cb }

// 35998c
c8 67 bool interseg(line r, line s) { // se o seg de r intersecta o
      seg de s
e3 19   if (isinseg(r.p, s) or isinseg(r.q, s)
14 c2      or isinseg(s.p, r) or isinseg(s.q, r)) return 1;

99 9f   return ccw(r.p, r.q, s.p) != ccw(r.p, r.q, s.q) and
40 41      ccw(s.p, s.q, r.p) != ccw(s.p, s.q, r.q);
a8 cb }

// 1b72e1
45 fc ld disttoline(pt p, line r) { // distancia do ponto a reta
8e 89   return 2 * abs(sarea(p, r.p, r.q)) / dist(r.p, r.q);
ad cb }

// 3679c0
4e bc ld disttoseg(pt p, line r) { // distancia do ponto ao seg
c3 73   if ((r.q - r.p)*(p - r.p) < 0) return dist(r.p, p);
f5 95   if ((r.p - r.q)*(p - r.q) < 0) return dist(r.q, p);
46 a1   return disttoline(p, r);
e3 cb }

// 222358
f9 11 ld distseg(line a, line b) { // distancia entre seg
48 4d   if (interseg(a, b)) return 0;

20 34   ld ret = DINF;
36 34   ret = min(ret, disttoseg(a.p, b));
14 ce   ret = min(ret, disttoseg(a.q, b));
4c 09   ret = min(ret, disttoseg(b.p, a));
67 44   ret = min(ret, disttoseg(b.q, a));

97 ed   return ret;
de cb }

// POLIGONO

// corta poligono com a reta r deixando os pontos p tal que
// ccw(r.p, r.q, p)
// 2538f9

```

```

9e 1a vector<pt> cut_polygon(vector<pt> v, line r) { // 0(n)
12 8a   vector<pt> ret;
f8 8a   for (int j = 0; j < v.size(); j++) {
89 da       if (ccw(r.p, r.q, v[j])) ret.push_back(v[j]);
49 dc       if (v.size() == 1) continue;
bd 03       line s(v[j], v[(j+1)%v.size()]);
fd ae       pt p = inter(r, s);
e3 a3       if (isinseg(p, s)) ret.push_back(p);
92 cb   }
d4 8a   ret.erase(unique(ret.begin(), ret.end()), ret.end());
ee 24   if (ret.size() > 1 and ret.back() == ret[0]) ret.pop_back();
b3 ed   return ret;
ec cb }

// distancia entre os retangulos a e b (lados paralelos aos eixos)
// assume que ta representado (inferior esquerdo, superior direito)
// 630253
86 5f ld dist_rect(pair<pt, pt> a, pair<pt, pt> b) {
7a 08   ld hor = 0, vert = 0;
c3 34   if (a.second.x < b.first.x) hor = b.first.x - a.second.x;
ab f5   else if (b.second.x < a.first.x) hor = a.first.x - b.second.x;
b8 4f   if (a.second.y < b.first.y) vert = b.first.y - a.second.y;
3b 80   else if (b.second.y < a.first.y) vert = a.first.y - b.second.y;
50 96   return dist(pt(0, 0), pt(hor, vert));
54 cb }

// 5df9cf
24 13 ld polarea(vector<pt> v) { // area do poligono
38 9c   ld ret = 0;
28 c6   for (int i = 0; i < v.size(); i++)
88 80       ret += sarea(pt(0, 0), v[i], v[(i + 1) % v.size()]);
a9 d0   return abs(ret);
e9 cb }

// se o ponto ta dentro do poligono: retorna 0 se ta fora,
// 1 se ta no interior e 2 se ta na borda
// a6423f
92 8e int inpol(vector<pt>& v, pt p) { // 0(n)
ef 8d   int qt = 0;
fc f1   for (int i = 0; i < v.size(); i++) {
7e bd       if (p == v[i]) return 2;
ea 6a       int j = (i+1)%v.size();
65 e3       if (eq(p.y, v[i].y) and eq(p.y, v[j].y)) {
6a 97           if ((v[i]-p)*(v[j]-p) < eps) return 2;
9d 5e           continue;
07 cb       }
14 38       bool baixo = v[i].y+eps < p.y;

```

```

76 46         if (baixo == (v[j].y+eps < p.y)) continue;
44 36         auto t = (p-v[i])^(v[j]-v[i]);
57 1b         if (eq(t, 0)) return 2;
8b 83         if (baixo == (t > eps)) qt += baixo ? 1 : -1;
7f cb     }
06 b8     return qt != 0;
62 cb }

// c58350
09 6f bool interpol(vector<pt> v1, vector<pt> v2) { // se dois
    poligonos se intersectam - O(n*m)
d7 7d     int n = v1.size(), m = v2.size();
51 c3     for (int i = 0; i < n; i++) if (inpol(v2, v1[i])) return 1;
7d ab     for (int i = 0; i < n; i++) if (inpol(v1, v2[i])) return 1;
05 52     for (int i = 0; i < n; i++) for (int j = 0; j < m; j++)
cc 0c         if (interseg(line(v1[i], v1[(i+1)%n]), line(v2[j],
    v2[(j+1)%m]))) return 1;
2e bb     return 0;
eb cb }

// 12559f
d7 49 ld distpol(vector<pt> v1, vector<pt> v2) { // distancia entre
    poligonos
ab f6     if (interpol(v1, v2)) return 0;

41 34     ld ret = DINF;

17 1c     for (int i = 0; i < v1.size(); i++) for (int j = 0; j <
    v2.size(); j++)
e7 6c         ret = min(ret, distseg(line(v1[i], v1[(i + 1) %
    v1.size()]),
69 9d             line(v2[j], v2[(j + 1) % v2.size()])));
9a ed     return ret;
55 cb }

// 10d7e0
f1 13 vector<pt> convex_hull(vector<pt> v) { // convex hull - O(n
    log(n))
8f fc     sort(v.begin(), v.end());
e8 d7     v.erase(unique(v.begin(), v.end()), v.end());
72 52     if (v.size() <= 1) return v;
0e 52     vector<pt> l, u;
57 f1     for (int i = 0; i < v.size(); i++) {
94 fb         while (l.size() > 1 and !ccw(l.end()[-2], l.end()[-1],
    v[i]))
31 36             l.pop_back();
77 c3         l.push_back(v[i]);

```

```

28 cb     }
9b 3e     for (int i = v.size() - 1; i >= 0; i--) {
d7 f1         while (u.size() > 1 and !ccw(u.end()[-2], u.end()[-1],
    v[i]))
37 7a             u.pop_back();
ff a9             u.push_back(v[i]);
35 cb     }
f9 cf     l.pop_back(); u.pop_back();
cf 82     for (pt i : u) l.push_back(i);
5d 79     return l;
25 cb }

36 48 struct convex_pol {
1f f5     vector<pt> pol;

    // nao pode ter ponto colinear no convex hull
8d d9     convex_pol() {}
06 a0     convex_pol(vector<pt> v) : pol(convex_hull(v)) {}

    // se o ponto ta dentro do hull - O(log(n))
    // 800813
f1 8a     bool is_inside(pt p) {
4f ea         if (pol.size() == 1) return p == pol[0];
19 67         int l = 1, r = pol.size();
30 40         while (l < r) {
2d ee             int m = (l+r)/2;
41 48             if (ccw(p, pol[0], pol[m])) l = m+1;
cb ef             else r = m;
7d cb         }
3b 00         if (l == 1) return isinseg(p, line(pol[0], pol[1]));
5d 9e         if (l == pol.size()) return false;
90 1c         return !ccw(p, pol[l], pol[l-1]);
3c cb     }

    // ponto extremo em relacao a cmp(p, q) = p mais extremo q
    // (copiado de https://github.com/gustavoM32/caderno-zika)
    // 56ccd2
49 71     int extreme(const function<bool(pt, pt)>& cmp) {
b5 b1         int n = pol.size();
2a 4a         auto extr = [&](int i, bool& cur_dir) {
e1 22             cur_dir = cmp(pol[(i+1)%n], pol[i]);
cd 61             return !cur_dir and !cmp(pol[(i+n-1)%n], pol[i]);
7f 21         };
46 63         bool last_dir, cur_dir;
aa a0         if (extr(0, last_dir)) return 0;
d3 99         int l = 0, r = n;
52 ea         while (l+1 < r) {
5c ee             int m = (l+r)/2;

```

```

83 f2         if (extr(m, cur_dir)) return m;
ae 44         bool rel_dir = cmp(pol[m], pol[l]);
da b1         if ((!last_dir and cur_dir) or
2c 26             (last_dir == cur_dir and rel_dir == cur_dir)) {
24 8a             l = m;
fd 1f             last_dir = cur_dir;
1f b6         } else r = m;
f2 cb     }
1d 79     return l;
cc cb     }
02 31     int max_dot(pt v) {
8e ec         return extreme([&](pt p, pt q) { return p*v > q*v; });
93 cb     }
33 a5     pair<int, int> tangents(pt p) {
cc ff         auto L = [&](pt q, pt r) { return ccw(p, r, q); };
0a 8f         auto R = [&](pt q, pt r) { return ccw(p, q, r); };
0c fa         return {extreme(L), extreme(R)};
76 cb     }
9e 21 };

// CIRCUNFERENCIA

// a125e4
54 91 pt getcenter(pt a, pt b, pt c) { // centro da circunf dado 3
    pontos
e2 17     b = (a + b) / 2;
a1 2a     c = (a + c) / 2;
5f 98     return inter(line(b, b + rotate90(a - b)),
6c 3f         line(c, c + rotate90(a - c)));
85 cb }

// cd80c0
8b 4b vector<pt> circ_line_inter(pt a, pt b, pt c, ld r) { //
    intersecao da circunf (c, r) e reta ab
4d 8a     vector<pt> ret;
63 f2     b = b-a, a = a-c;
3b 4b     ld A = b*b;
a4 20     ld B = a*b;
dd 2e     ld C = a*a - r*r;
87 1f     ld D = B*B - A*C;
1b 81     if (D < -eps) return ret;
26 dc     ret.push_back(c+a+b*(-B+sqrt(D+eps))/A);
3b 20     if (D > eps) ret.push_back(c+a+b*(-B-sqrt(D))/A);
45 ed     return ret;
59 cb }

// fb11d8

```

```

a4 ad vector<pt> circ_inter(pt a, pt b, ld r, ld R) { // intersecao da
    circunf (a, r) e (b, R)
fd 8a     vector<pt> ret;
45 b7     ld d = dist(a, b);
67 5c     if (d > r+R or d+min(r, R) < max(r, R)) return ret;
69 39     ld x = (d*d-R*R+r*r)/(2*d);
15 18     ld y = sqrt(r*r-x*x);
f3 32     pt v = (b-a)/d;
41 76     ret.push_back(a+v*x + rotate90(v)*y);
8e 2c     if (y > 0) ret.push_back(a+v*x - rotate90(v)*y);
2b ed     return ret;
e9 cb }

// 3a44fb
07 6e bool operator <(const line& a, const line& b) { // comparador
    pra reta
        // assume que as retas tem p < q
bb a1     pt v1 = a.q - a.p, v2 = b.q - b.p;
42 f8     if (!eq(angle(v1), angle(v2))) return angle(v1) < angle(v2);
11 78     return ccw(a.p, a.q, b.p); // mesmo angulo
0c cb }
53 b1 bool operator ==(const line& a, const line& b) {
d6 76     return !(a < b) and !(b < a);
bf cb }

// comparador pro set pra fazer sweep line com segmentos
// 36729f
2a 2c struct cmp_sweepline {
5b d8     bool operator () (const line& a, const line& b) const {
        // assume que os segmentos tem p < q
1e 19         if (a.p == b.p) return ccw(a.p, a.q, b.q);
79 23         if (!eq(a.p.x, a.q.x) and (eq(b.p.x, b.q.x) or a.p.x+eps <
            b.p.x))
49 78             return ccw(a.p, a.q, b.p);
25 dc         return ccw(a.p, b.q, b.p);
4e cb     }
60 21 };

// comparador pro set pra fazer sweep angle com segmentos
// f778aa
b4 be pt dir;
b1 5b struct cmp_sweepangle {
87 d8     bool operator () (const line& a, const line& b) const {
b2 52         return get_t(dir, a) + eps < get_t(dir, b);
8d cb     }
49 21 };

```

## 5.7 Primitivas Geometricas 3D

```

c8 c8 typedef double ld;
a4 e3 const ld DINF = 1e18;
4b 10 const ld eps = 1e-9;

fc b3 #define sq(x) ((x)*(x))

fe d9 bool eq(ld a, ld b) {
10 ba     return abs(a - b) <= eps;
7e cb }

// 3eef01
ce b2 struct pt { // ponto
c6 2e     ld x, y, z;
be a5     pt(ld x_ = 0, ld y_ = 0, ld z_ = 0) : x(x_), y(y_),
        z(z_) {}
62 5b     bool operator < (const pt p) const {
51 05         if (!eq(x, p.x)) return x < p.x;
e5 f9         if (!eq(y, p.y)) return y < p.y;
29 44         if (!eq(z, p.z)) return z < p.z;
cf bb         return 0;
3e cb     }
a2 a8     bool operator == (const pt p) const {
4d 41         return eq(x, p.x) and eq(y, p.y) and eq(z, p.z);
85 cb     }
cc 44     pt operator + (const pt p) const { return pt(x+p.x,
        y+p.y, z+p.z); }
84 39     pt operator - (const pt p) const { return pt(x-p.x,
        y-p.y, z-p.z); }
45 fb     pt operator * (const ld c) const { return pt(x*c , y*c
        , z*c ); }
ac 7a     pt operator / (const ld c) const { return pt(x/c , y/c
        , z/c ); }
06 a6     ld operator * (const pt p) const { return x*p.x + y*p.y
        + z*p.z; }
09 7f     pt operator ^ (const pt p) const { return pt(y*p.z -
        z*p.y, z*p.x - x*p.z, x*p.y - y*p.x); }
b9 5e     friend istream& operator >> (istream& in, pt& p) {
9e 9b         return in >> p.x >> p.y >> p.z;
31 cb     }
83 21 };

// 7ab617
02 b3 struct line { // reta
aa 73     pt p, q;
6c 0d     line() {}

```

```

db 4b     line(pt p_, pt q_) : p(p_), q(q_) {}
ff 8d     friend istream& operator >> (istream& in, line& r) {
9b 4c         return in >> r.p >> r.q;
25 cb     }
00 21 };

// d5d580
e7 79 struct plane { // plano
15 7e     array<pt, 3> p; // pontos que definem o plano
b0 29     array<ld, 4> eq; // equacao do plano
d6 bb     plane() {}
6a fb     plane(pt p_, pt q_, pt r_) : p({p_, q_, r_}) { build(); }

cd ca     friend istream& operator >> (istream& in, plane& P) {
02 2a         return in >> P.p[0] >> P.p[1] >> P.p[2];
7b 70         P.build();
56 cb     }
57 0a     void build() {
6a da         pt dir = (p[1] - p[0]) ^ (p[2] - p[0]);
3f 7d         eq = {dir.x, dir.y, dir.z, dir*p[0]*(-1)};
03 cb     }
54 21 };

// converte de coordenadas polares para cartesianas
// (angulos devem estar em radianos)
// phi eh o angulo com o eixo z (cima) theta eh o angulo de rotacao ao
// redor de z
// a4f17f
2e 2f pt convert(ld rho, ld th, ld phi) {
a7 cf     return pt(sin(phi) * cos(th), sin(phi) * sin(th),
ed cb     cos(phi)) * rho;

// projecao do ponto p na reta r
// 2329fe
ec 25 pt proj(pt p, line r) {
6e be     if (r.p == r.q) return r.p;
3c 97     r.q = r.q - r.p; p = p - r.p;
8c 9f     pt proj = r.q * ((p*r.q) / (r.q*r.q));
21 2c     return proj + r.p;
be cb }

// projecao do ponto p no plano P
// 4a0d14
59 b1 pt proj(pt p, plane P) {
dc 7b     p = p - P.p[0], P.p[1] = P.p[1] - P.p[0], P.p[2] =
        P.p[2] - P.p[0];

```

```

3e b6      pt norm = P.p[1] ^ P.p[2];
05 6a      pt proj = p - (norm * (norm * p) / (norm*norm));
5c 46      return proj + P.p[0];
c7 cb }

// distancia
// 2d06b0
fd a4 ld dist(pt a, pt b) {
09 fd      return sqrt(sq(a.x-b.x) + sq(a.y-b.y) + sq(a.z-b.z));
ec cb }

// distancia ponto reta
// 3c4e1b
76 13 ld distline(pt p, line r) {
ef ce      return dist(p, proj(p, r));
d6 cb }

// distancia de ponto para segmento
// 42cbbd
df d4 ld distseg(pt p, line r) {
36 73      if ((r.q - r.p)*(p - r.p) < 0) return dist(r.p, p);
e1 95      if ((r.p - r.q)*(p - r.q) < 0) return dist(r.q, p);
d3 20      return distline(p, r);
6b cb }

// distancia de ponto a plano com sinal
// d490d9
f9 7c ld sdist(pt p, plane P) {
1f 15      return P.eq[0]*p.x + P.eq[1]*p.y + P.eq[2]*p.z +
P.eq[3];
b9 cb }

// distancia de ponto a plano
// 33dc8c
cc 76 ld distplane(pt p, plane P) {
73 c3      return abs(sdist(p, P));
95 cb }

// se ponto pertence a reta
// 31a295
71 09 bool isinseg(pt p, line r) {
93 a3      return eq(distseg(p, r), 0);
45 cb }

// se ponto pertence ao triangulo definido por P.p
// c81f7e
34 cd bool isinpol(pt p, vector<pt> v) {

```

```

78 fa      assert(v.size() >= 3);
62 bf      pt norm = (v[1]-v[0]) ^ (v[2]-v[1]);
98 8a      bool inside = true;
24 ce      int sign = -1;
ab f1      for (int i = 0; i < v.size(); i++) {
a6 83          line r(v[(i+1)%3], v[i]);
6a 2a          if (isinseg(p, r)) return true;

7b 4e          pt ar = v[(i+1)%3] - v[i];
0f 32          if (sign == -1) sign = ((ar^(p-v[i]))*norm > 0);
f4 82          else if (((ar^(p-v[i]))*norm > 0) != sign)
inside = false;
d3 cb      }
0a ac      return inside;
40 cb }

// distancia de ponto ate poligono
// a8d4c2
1b 36 ld distpol(pt p, vector<pt> v) {
2f 3e      pt p2 = proj(p, plane(v[0], v[1], v[2]));
4c 61      if (isinpol(p2, v)) return dist(p, p2);
b8 34      ld ret = DINF;
d6 f1      for (int i = 0; i < v.size(); i++) {
ba 6a          int j = (i+1)%v.size();
9d 5e          ret = min(ret, distseg(p, line(v[i], v[j])));
1f cb      }
4e ed      return ret;
b1 cb }

// intersecao de plano e segmento
// BOTH = o segmento esta no plano
// ONE = um dos pontos do segmento esta no plano
// PARAL = segmento paralelo ao plano
// CONCOR = segmento concorrente ao plano
// e2ecac
e8 e5 enum RETCODE {BOTH, ONE, PARAL, CONCOR};
2f 26 pair<RETCODE, pt> intersect(plane P, line r) {
6b fa      ld d1 = sdist(r.p, P);
af f8      ld d2 = sdist(r.q, P);
bb 53      if (eq(d1, 0) and eq(d2, 0))
21 50          return pair(BOTH, r.p);
a4 72      if (eq(d1, 0))
48 84          return pair(ONE, r.p);
e3 48      if (eq(d2, 0))
3a 16          return pair(ONE, r.q);
4d 3f      if ((d1 > 0 and d2 > 0) or (d1 < 0 and d2 < 0)) {
7a 46          if (eq(d1-d2, 0)) return pair(PARAL, pt());

```



```

ab 40         return pair(CONCOR, pt());
6a cb      }
0b c8      ld frac = d1 / (d1 - d2);
9a 3f      pt res = r.p + ((r.q - r.p) * frac);
5e 39      return pair(ONE, res);
37 cb }

// rotaciona p ao redor do eixo u por um angulo a
// 7f0a40
07 78 pt rotate(pt p, pt u, ld a) {
cc 77      u = u / dist(u, pt());
92 e6      return u * (u * p) + (u ^ p ^ u) * cos(a) + (u ^ p) *
          sin(a);
11 cb }

```

## 5.8 Primitivas Geometricas Inteiras

```

2d 2d #define sq(x) ((x)*(1l)(x))

// 840720
61 b2 struct pt { // ponto
82 e9     int x, y;
a6 df     pt(int x_ = 0, int y_ = 0) : x(x_), y(y_) {}
61 5b     bool operator < (const pt p) const {
b3 95         if (x != p.x) return x < p.x;
51 89         return y < p.y;
14 cb     }
f4 a8     bool operator == (const pt p) const {
59 d7         return x == p.x and y == p.y;
5e cb     }
b0 cb     pt operator + (const pt p) const { return pt(x+p.x, y+p.y); }
e3 a2     pt operator - (const pt p) const { return pt(x-p.x, y-p.y); }
29 0e     pt operator * (const int c) const { return pt(x*c, y*c); }
0f 60     ll operator * (const pt p) const { return x*(1l)p.x +
          y*(1l)p.y; }
63 d8     ll operator ^ (const pt p) const { return x*(1l)p.y -
          y*(1l)p.x; }
b1 5e     friend istream& operator >> (istream& in, pt& p) {
e3 e3         return in >> p.x >> p.y;
48 cb     }
af 21 };

// 7ab617
d8 b3 struct line { // reta
53 73     pt p, q;
07 0d     line() {}
00 4b     line(pt p_, pt q_) : p(p_), q(q_) {}

```

```

e7 8d     friend istream& operator >> (istream& in, line& r) {
0f 4c         return in >> r.p >> r.q;
60 cb     }
b1 21 };

// PONTO & VETOR

// 51563e
2b ea ll dist2(pt p, pt q) { // quadrado da distancia
70 f2     return sq(p.x - q.x) + sq(p.y - q.y);
bc cb }

// bf431d
8c 5a ll sarea2(pt p, pt q, pt r) { // 2 * area com sinal
0c 58     return (q-p)^(r-q);
9f cb }

// a082d3
c1 e3 bool col(pt p, pt q, pt r) { // se p, q e r sao colin.
09 03     return sarea2(p, q, r) == 0;
8b cb }

// 42bb09
fd 0c bool ccw(pt p, pt q, pt r) { // se p, q, r sao ccw
6a 27     return sarea2(p, q, r) > 0;
d3 cb }

// fcf924
9c c3 int quad(pt p) { // quadrante de um ponto
a9 db     return (p.x<0)^3*(p.y<0);
27 cb }

// 77187b
39 2d bool compare_angle(pt p, pt q) { // retorna se ang(p) < ang(q)
e9 9f     if (quad(p) != quad(q)) return quad(p) < quad(q);
3d ea     return ccw(q, pt(0, 0), p);
2b cb }

// e4ad5e
23 ab pt rotate90(pt p) { // rotaciona 90 graus
4c a0     return pt(-p.y, p.x);
37 cb }

// RETA

// c9f07f
de 09 bool isinseg(pt p, line r) { // se p pertence ao seg de r

```

```

8c f6 pt a = r.p - p, b = r.q - p;
5e 2a return (a ^ b) == 0 and (a * b) <= 0;
3d cb }

// 35998c
38 67 bool interseg(line r, line s) { // se o seg de r intersecta o
    seg de s
e3 19 if (isinseg(r.p, s) or isinseg(r.q, s)
6e c2 or isinseg(s.p, r) or isinseg(s.q, r)) return 1;

5b 9f return ccw(r.p, r.q, s.p) != ccw(r.p, r.q, s.q) and
8e 41 ccw(s.p, s.q, r.p) != ccw(s.p, s.q, r.q);
47 cb }

// dd8702
e7 9e int segpoints(line r) { // numero de pontos inteiros no segmento
12 9c return 1 + __gcd(abs(r.p.x - r.q.x), abs(r.p.y - r.q.y));
6d cb }

// d273be
f4 88 double get_t(pt v, line r) { // retorna t tal que t*v pertence a
    reta r
5f 1a return (r.p^r.q) / (double) ((r.p-r.q)^v);
df cb }

// POLIGONO

// quadrado da distancia entre os retangulos a e b (lados paralelos
    aos eixos)
// assume que ta representado (inferior esquerdo, superior direito)
// e13018
eb 48 ll dist2_rect(pair<pt, pt> a, pair<pt, pt> b) {
73 c5 int hor = 0, vert = 0;
80 34 if (a.second.x < b.first.x) hor = b.first.x - a.second.x;
47 f5 else if (b.second.x < a.first.x) hor = a.first.x - b.second.x;
b0 4f if (a.second.y < b.first.y) vert = b.first.y - a.second.y;
5b 80 else if (b.second.y < a.first.y) vert = a.first.y - b.second.y;
f1 86 return sq(hor) + sq(vert);
ee cb }

// d5f693
d5 9c ll polarea2(vector<pt> v) { // 2 * area do poligono
aa b7 ll ret = 0;
68 c6 for (int i = 0; i < v.size(); i++)
d1 53 ret += sarea2(pt(0, 0), v[i], v[(i + 1) % v.size()]);
f1 d0 return abs(ret);
74 cb }

```

```

// se o ponto ta dentro do poligono: retorna 0 se ta fora,
// 1 se ta no interior e 2 se ta na borda
// afd587
a2 8e int inpol(vector<pt>& v, pt p) { // 0(n)
ba 8d int qt = 0;
e9 f1 for (int i = 0; i < v.size(); i++) {
84 bd if (p == v[i]) return 2;
77 6a int j = (i+1)%v.size();
e0 cc if (p.y == v[i].y and p.y == v[j].y) {
9d 54 if ((v[i]-p)*(v[j]-p) <= 0) return 2;
86 5e continue;
c8 cb }
d4 78 bool baixo = v[i].y < p.y;
48 05 if (baixo == (v[j].y < p.y)) continue;
10 36 auto t = (p-v[i])^(v[j]-v[i]);
bb 2a if (!t) return 2;
cb 0b if (baixo == (t > 0)) qt += baixo ? 1 : -1;
f2 cb }
b8 b8 return qt != 0;
28 cb }

// 10d7e0
bf 13 vector<pt> convex_hull(vector<pt> v) { // convex hull - 0(n
    log(n))
f7 fc sort(v.begin(), v.end());
a6 d7 v.erase(unique(v.begin(), v.end()), v.end());
55 52 if (v.size() <= 1) return v;
45 52 vector<pt> l, u;
bf f1 for (int i = 0; i < v.size(); i++) {
05 fb while (l.size() > 1 and !ccw(l.end()[-2], l.end()[-1],
    v[i]))
f7 36 l.pop_back();
0d c3 l.push_back(v[i]);
36 cb }
18 3e for (int i = v.size() - 1; i >= 0; i--) {
1f f1 while (u.size() > 1 and !ccw(u.end()[-2], u.end()[-1],
    v[i]))
6e 7a u.pop_back();
0b a9 u.push_back(v[i]);
d3 cb }
5e cf l.pop_back(); u.pop_back();
88 82 for (pt i : u) l.push_back(i);
eb 79 return l;
6b cb }

// af2d96

```

```

9e 78 ll interior_points(vector<pt> v) { // pontos inteiros dentro de
    um poligono simples
94 c4    ll b = 0;
e6 c6    for (int i = 0; i < v.size(); i++)
74 0c        b += segpoints(line(v[i], v[(i+1)%v.size()])) - 1;
9c a1    return (polarea2(v) - b) / 2 + 1;
51 cb }

71 48 struct convex_pol {
2d f5    vector<pt> pol;

    // nao pode ter ponto colinear no convex hull
b3 d9    convex_pol() {}
bf a0    convex_pol(vector<pt> v) : pol(convex_hull(v)) {}

    // se o ponto ta dentro do hull - O(log(n))
    // 800813
4d 8a    bool is_inside(pt p) {
dc ea        if (pol.size() == 1) return p == pol[0];
d4 67        int l = 1, r = pol.size();
3b 40        while (l < r) {
4e ee            int m = (l+r)/2;
05 48            if (ccw(p, pol[0], pol[m])) l = m+1;
c1 ef            else r = m;
c6 cb        }
c4 00        if (l == 1) return isinseg(p, line(pol[0], pol[1]));
fc 9e        if (l == pol.size()) return false;
5d 1c        return !ccw(p, pol[l], pol[l-1]);
08 cb    }

    // ponto extremo em relacao a cmp(p, q) = p mais extremo q
    // (copiado de https://github.com/gustavoM32/caderno-zika)
    // 56ccd2
ef 71    int extreme(const function<bool(pt, pt)>& cmp) {
de b1        int n = pol.size();
11 4a        auto extr = [&](int i, bool& cur_dir) {
6b 22            cur_dir = cmp(pol[(i+1)%n], pol[i]);
62 61            return !cur_dir and !cmp(pol[(i+n-1)%n], pol[i]);
72 21        };
3b 63        bool last_dir, cur_dir;
ee a0        if (extr(0, last_dir)) return 0;
41 99        int l = 0, r = n;
66 ea        while (l+1 < r) {
60 ee            int m = (l+r)/2;
1b f2            if (extr(m, cur_dir)) return m;
49 44            bool rel_dir = cmp(pol[m], pol[l]);
d6 b1            if ((!last_dir and cur_dir) or
b6 26                (last_dir == cur_dir and rel_dir == cur_dir)) {

```

```

df 8a                l = m;
ab 1f                last_dir = cur_dir;
d0 b6                } else r = m;
fd cb            }
77 79            return l;
76 cb        }
ad 31    int max_dot(pt v) {
39 ec        return extreme([&](pt p, pt q) { return p*v > q*v; });
18 cb    }
11 a5    pair<int, int> tangents(pt p) {
f9 ff        auto L = [&](pt q, pt r) { return ccw(p, r, q); };
47 8f        auto R = [&](pt q, pt r) { return ccw(p, q, r); };
c3 fa        return {extreme(L), extreme(R)};
32 cb    }
f2 21 };

// dca598
82 6e bool operator <(const line& a, const line& b) { // comparador
    pra reta
        // assume que as retas tem p < q
3d a1    pt v1 = a.q - a.p, v2 = b.q - b.p;
0f 03    bool b1 = compare_angle(v1, v2), b2 = compare_angle(v2, v1);
a4 73    if (b1 or b2) return b1;
5d 78    return ccw(a.p, a.q, b.p); // mesmo angulo
db cb }

4f b1 bool operator ==(const line& a, const line& b) {
fe 76    return !(a < b) and !(b < a);
22 cb }

// comparador pro set pra fazer sweep line com segmentos
// 6774df
ad 2c struct cmp_sweepline {
4b d8    bool operator () (const line& a, const line& b) const {
        // assume que os segmentos tem p < q
0f 19        if (a.p == b.p) return ccw(a.p, a.q, b.q);
20 61        if (a.p.x != a.q.x and (b.p.x == b.q.x or a.p.x < b.p.x))
28 78            return ccw(a.p, a.q, b.p);
d9 dc        return ccw(a.p, b.q, b.p);
40 cb    }
8e 21 };

// comparador pro set pra fazer sweep angle com segmentos
// 1ee7f5
d3 be pt dir;
dd 5b struct cmp_sweepangle {
45 d8    bool operator () (const line& a, const line& b) const {
5c 26        return get_t(dir, a) < get_t(dir, b);

```

```

21 cb      }
8a 21 };

```

## 6 Estruturas

### 6.1 BIT

```

// BIT de soma 1-based, v 0-based
// Para mudar o valor da posicao p para x,
// faca: poe(x - query(p, p), p)
// l_bound(x) retorna o menor p tal que
// query(1, p+1) > x      (0 based!)
//
// Complexidades:
// build - O(n)
// poe - O(log(n))
// query - O(log(n))
// l_bound - O(log(n))
// d432a4

1a 1a int n;
bc 7f int bit[MAX];
a2 b6 int v[MAX];

d5 0a void build() {
a5 b9     bit[0] = 0;
e5 33     for (int i = 1; i <= n; i++) bit[i] = v[i - 1];

97 78     for (int i = 1; i <= n; i++) {
9e ed         int j = i + (i & -i);
2f b8         if (j <= n) bit[j] += bit[i];
f5 cb     }
41 cb }

// soma x na posicao p
b4 23 void poe(int x, int p) {
19 9c     for (; p <= n; p += p & -p) bit[p] += x;
c7 cb }

// soma [1, p]
72 0b int pref(int p) {
44 7c     int ret = 0;
50 80     for (; p; p -= p & -p) ret += bit[p];
7f ed     return ret;
2f cb }

```

```

// soma [a, b]
1c 4e int query(int a, int b) {
31 70     return pref(b) - pref(a - 1);
83 cb }

ff e4 int l_bound(ll x) {
c7 1b     int p = 0;
34 67     for (int i = MAX2; i+1; i--) if (p + (1<<i) <= n
cf 72         and bit[p + (1<<i)] <= x) x -= bit[p += (1<<i)];
1f 74     return p;
d4 cb }

```

### 6.2 BIT 2D

```

// BIT de soma, update incrementa posicao
// Tem que construir com um vetor com todos os pontos
// que vc quer um dia atualizar (os pontos q vc vai chamar update)
//
// Complexidades:
// construir - O(n log(n))
// update e query - O(log^2(n))
// 6a760a

a6 a6 template<class T = int> struct bit2d {
1b ac     vector<T> X;
03 a8     vector<vector<T>> Y, t;

fa 70     int ub(vector<T>& v, T x) {
39 dd         return upper_bound(v.begin(), v.end(), x) - v.begin();
b6 cb     }
99 5c     bit2d(vector<pair<T, T>> v) {
04 2e         for (auto [x, y] : v) X.push_back(x);
75 fd         sort(X.begin(), X.end());
da 1e         X.erase(unique(X.begin(), X.end()), X.end());

e3 d5         t.resize(X.size() + 1);
27 d1         Y.resize(t.size());
c5 3d         sort(v.begin(), v.end(), [](auto a, auto b) {
ff 43             return a.second < b.second; });
a3 96         for (auto [x, y] : v) for (int i = ub(X, x); i < t.size();
i += i&-i)
38 b7             if (!Y[i].size() or Y[i].back() != y)
Y[i].push_back(y);

7b 7c         for (int i = 0; i < t.size(); i++) t[i].resize(Y[i].size()
+ 1);
ab cb     }

```

```

ed e7 void update(T x, T y, T v) {
aa 2a     for (int i = ub(X, x); i < t.size(); i += i&-i)
cd cd         for (int j = ub(Y[i], y); j < t[i].size(); j += j&-j)
            t[i][j] += v;
fb cb     }

b0 5d T query(T x, T y) {
7e 96     T ans = 0;
36 c5     for (int i = ub(X, x); i; i -= i&-i)
9d 4f         for (int j = ub(Y[i], y); j; j -= j&-j) ans += t[i][j];
e5 ba     return ans;
3b cb     }
1a 46 T query(T x1, T y1, T x2, T y2) {
00 fc     return query(x2, y2)-query(x2, y1-1)-query(x1-1,
            y2)+query(x1-1, y1-1);
1d cb     }
6a 21 };

```

### 6.3 BIT com update em range

```

// Operacoes 0-based
// query(l, r) retorna a soma de v[l..r]
// update(l, r, x) soma x em v[l..r]
//
// Complexidades:
// build - O(n)
// query - O(log(n))
// update - O(log(n))
// f91737

```

```

e0 e0 namespace bit {
8b 3b     ll bit[2][MAX+2];
ab 1a     int n;

f2 61 void build(int n2, int* v) {
7b 1e     n = n2;
35 53     for (int i = 1; i <= n; i++)
55 ed         bit[1][min(n+1, i+(i&-i))] += bit[1][i] += v[i-1];
30 cb     }
00 63 ll get(int x, int i) {
43 b7     ll ret = 0;
7f 36     for (; i; i -= i&-i) ret += bit[x][i];
3e ed     return ret;
85 cb     }
41 20 void add(int x, int i, ll val) {
fe 50     for (; i <= n; i += i&-i) bit[x][i] += val;

```

```

fa cb     }
4e 16 ll get2(int p) {
cd c7     return get(0, p) * p + get(1, p);
3e cb     }
41 02 ll query(int l, int r) {
2e ff     return get2(r+1) - get2(l);
f5 cb     }
34 08 void update(int l, int r, ll x) {
b2 e5     add(0, l+1, x), add(0, r+2, -x);
ac f5     add(1, l+1, -x*1), add(1, r+2, x*(r+1));
4e cb     }
f9 21 };

```

### 6.4 BIT-Sort Tree

```

// Tipo uma MergeSort Tree usando Bit
// Apesar da complexidade ser pior, fica melhor na pratica.
//
// query(l, r, k) retorna o numero de elementos menores que k
// no intervalo [l, r]
//
// Usa O(n log(n)) de memoria
//
// Complexidades:
// construir - O(n log^2(n))
// query - O(log^2(n))
// 8d0749

```

```

6f 6f template<typename T> struct ms_bit {
54 1a     int n;
44 b2     vector<vector<T>> bit;

95 89     ms_bit(vector<T>& v) : n(v.size()), bit(n+1) {
20 83         for (int i = 0; i < n; i++)
6d d5             for (int j = i+1; j <= n; j += j&-j)
05 da                 bit[j].push_back(v[i]);
1d 53         for (int i = 1; i <= n; i++)
8e ee             sort(bit[i].begin(), bit[i].end());
f4 cb     }

72 25     int p_query(int i, T k) {
96 7c         int ret = 0;
67 be         for (i++; i; i -= i&-i)
af 1b             ret += lower_bound(bit[i].begin(), bit[i].end(), k)
                - bit[i].begin();
7b ed         return ret;
97 cb     }

```

```

56 69     int query(int l, int r, T k) {
96 83         return p_query(r, k) - p_query(l-1, k);
32 cb     }
8d 21 };

```

## 6.5 DSU

```

// Une dois conjuntos e acha a qual conjunto um elemento pertence por
// seu id
//
// find e unite:  $O(a(n)) \sim O(1)$  amortizado
// 8e197e

8d 8d struct dsu {
b7 82     vector<int> id, sz;

0e b3     dsu(int n) : id(n), sz(n, 1) { iota(id.begin(), id.end(), 0); }

cf 0c     int find(int a) { return a == id[a] ? a : id[a] = find(id[a]);
    }

c7 44     void unite(int a, int b) {
ad 60         a = find(a), b = find(b);
1f d5         if (a == b) return;
b6 95         if (sz[a] < sz[b]) swap(a, b);
81 6d         sz[a] += sz[b], id[b] = a;
52 cb     }
8e 21 };

// DSU de bipartido
//
// Une dois vertices e acha a qual componente um vertice pertence
// Informa se a componente de um vertice e bipartida
//
// find e unite:  $O(\log(n))$ 
// 118050

dc 8d struct dsu {
4a 6f     vector<int> id, sz, bip, c;

8c 5b     dsu(int n) : id(n), sz(n, 1), bip(n, 1), c(n) {
0f db         iota(id.begin(), id.end(), 0);
2e cb     }

b4 ef     int find(int a) { return a == id[a] ? a : find(id[a]); }
04 f3     int color(int a) { return a == id[a] ? c[a] : c[a] =
    color(id[a]); }

```

```

74 44     void unite(int a, int b) {
f9 26         bool change = color(a) == color(b);
04 60         a = find(a), b = find(b);
fe a8         if (a == b) {
b5 4e             if (change) bip[a] = 0;
b7 50             return;
4b cb         }

17 95         if (sz[a] < sz[b]) swap(a, b);
3b ef         if (change) c[b] = 1;
41 2c         sz[a] += sz[b], id[b] = a, bip[a] &= bip[b];
6b cb     }
f0 21 };

```

```

// DSU Persistente
//
// Persistencia parcial, ou seja, tem que ir
// incrementando o 't' no une
//
// find e unite:  $O(\log(n))$ 
// 6c63a4

```

```

34 8d struct dsu {
d4 33     vector<int> id, sz, ti;

53 73     dsu(int n) : id(n), sz(n, 1), ti(n, -INF) {
3f db         iota(id.begin(), id.end(), 0);
6a cb     }

52 5e     int find(int a, int t) {
2e 6b         if (id[a] == a or ti[a] > t) return a;
46 ea         return find(id[a], t);
34 cb     }

b3 fa     void unite(int a, int b, int t) {
6b 84         a = find(a, t), b = find(b, t);
99 d5         if (a == b) return;
5e 95         if (sz[a] < sz[b]) swap(a, b);
43 35         sz[a] += sz[b], id[b] = a, ti[b] = t;
b5 cb     }
a9 21 };

```

```

// DSU com rollback
//
// checkpoint(): salva o estado atual de todas as variaveis

```

```

// rollback(): retorna para o valor das variaveis para
// o ultimo checkpoint
//
// Sempre que uma variavel muda de valor, adiciona na stack
//
// find e unite: O(log(n))
// checkpoint: O(1)
// rollback: O(m) em que m e o numero de vezes que alguma
// variavel mudou de valor desde o ultimo checkpoint
// c6e923

4f 8d struct dsu {
32 82     vector<int> id, sz;
7f 27     stack<stack<pair<int&, int>>> st;

21 98     dsu(int n) : id(n), sz(n, 1) {
9f 1c         iota(id.begin(), id.end(), 0), st.emplace();
4f cb     }

e9 bd     void save(int &x) { st.top().emplace(x, x); }

a2 30     void checkpoint() { st.emplace(); }

1b 5c     void rollback() {
67 ba         while(st.top().size()) {
08 6b             auto [end, val] = st.top().top(); st.top().pop();
1d 14             end = val;
86 cb         }
5c 25         st.pop();
ed cb     }

a7 ef     int find(int a) { return a == id[a] ? a : find(id[a]); }

b9 44     void unite(int a, int b) {
68 60         a = find(a), b = find(b);
8a d5         if (a == b) return;
92 95         if (sz[a] < sz[b]) swap(a, b);
a8 80         save(sz[a]), save(id[b]);
0e 6d         sz[a] += sz[b], id[b] = a;
5b cb     }
c4 21 };

```

## 6.6 Li-Chao Tree

```

// Adiciona retas (ax+b), e computa o minimo entre as retas
// em um dado 'x'
// Cuidado com overflow!

```

```

// Se tiver overflow, tenta comprimir o 'x' ou usar
// convex hull trick
//
// O(log(MA-MI)), O(n) de memoria
// 59ba68

```

```

5b 5b template<ll MI = ll(-1e9), ll MA = ll(1e9)> struct lichao {
23 b3     struct line {
6a 12         ll a, b;
a9 ce         array<int, 2> ch;
55 fd         line(ll a_ = 0, ll b_ = LINF) :
3f 42             a(a_), b(b_), ch({-1, -1}) {}
b0 88         ll operator ()(ll x) { return a*x + b; }
f7 21     };
de 17     vector<line> ln;

09 df     int ch(int p, int d) {
90 e8         if (ln[p].ch[d] == -1) {
a3 9a             ln[p].ch[d] = ln.size();
a7 cd             ln.emplace_back();
0a cb         }
eb ef         return ln[p].ch[d];
a4 cb     }
62 02     lichao() { ln.emplace_back(); }

8e c3     void add(line s, ll l=MI, ll r=MA, int p=0) {
e9 3e         ll m = (l+r)/2;
ed 91         bool L = s.l < ln[p].l;
ef d3         bool M = s.m < ln[p].m;
d2 03         bool R = s.r < ln[p].r;
30 82         if (M) swap(ln[p], s), swap(ln[p].ch, s.ch);
35 ca         if (s.b == LINF) return;
10 f6         if (L != M) add(s, l, m-1, ch(p, 0));
e6 89         else if (R != M) add(s, m+1, r, ch(p, 1));
76 cb     }

67 09     ll query(int x, ll l=MI, ll r=MA, int p=0) {
24 11         ll m = (l+r)/2, ret = ln[p].x;
db 9d         if (ret == LINF) return ret;
e0 52         if (x < m) return min(ret, query(x, l, m-1, ch(p, 0)));
6d 81         return min(ret, query(x, m+1, r, ch(p, 1)));
aa cb     }
59 21 };

```

## 6.7 MergeSort Tree

```

// Se for construida sobre um array:
//     count(i, j, a, b) retorna quantos

```

```

//      elementos de v[i..j] pertencem a [a, b]
//      report(i, j, a, b) retorna os indices dos
//      elementos de v[i..j] que pertencem a [a, b]
//      retorna o vetor ordenado
// Se for construida sobre pontos (x, y):
//      count(x1, x2, y1, x2) retorna quantos pontos
//      pertencem ao retangulo (x1, y1), (x2, y2)
//      report(x1, x2, y1, y2) retorna os indices dos pontos que
//      pertencem ao retangulo (x1, y1), (x2, y2)
//      retorna os pontos ordenados lexicograficamente
//      (assume x1 <= x2, y1 <= y2)
//
// kth(y1, y2, k) retorna o indice do ponto com k-esimo menor
// x dentre os pontos que possuem y em [y1, y2] (0 based)
// Se quiser usar para achar k-esimo valor em range, construir
// com ms_tree t(v, true), e chamar kth(l, r, k)
//
// Usa O(n log(n)) de memoria
//
// Complexidades:
// construir - O(n log(n))
// count - O(log(n))
// report - O(log(n) + k) para k indices retornados
// kth - O(log(n))
// 1cef03

c6 c6 template <typename T = int> struct ms_tree {
c4 6f     vector<tuple<T, T, int>>> v;
c7 1a     int n;
2a 5e     vector<vector<tuple<T, T, int>>>> t; // {y, idx, left}
c9 6a     vector<T> vy;

ba 78     ms_tree(vector<pair<T, T>>& vv) : n(vv.size()), t(4*n), vy(n) {
23 e8         for (int i = 0; i < n; i++) v.push_back({vv[i].first,
vv[i].second, i});
aa fc         sort(v.begin(), v.end());
84 22         build(1, 0, n-1);
86 01         for (int i = 0; i < n; i++) vy[i] = get<0>(t[1][i+1]);
2c cb     }
b7 da     ms_tree(vector<T>& vv, bool inv = false) { // inv: inverte
indice e valor
bb 8e         vector<pair<T, T>> v2;
4a e1         for (int i = 0; i < vv.size(); i++)
5f 19             inv ? v2.push_back({vv[i], i}) : v2.push_back({i,
vv[i]});
d7 cc         *this = ms_tree(v2);
0e cb     }

```

```

04 2c     void build(int p, int l, int r) {
ef 1d         t[p].push_back({get<0>(v[l]), get<0>(v[r]), 0}); //
{min_x, max_x, 0}
0c 5c         if (l == r) return t[p].push_back({get<1>(v[l]),
get<2>(v[l]), 0});
33 ee         int m = (l+r)/2;
2f bd         build(2*p, l, m), build(2*p+1, m+1, r);

19 32         int L = 0, R = 0;
84 a0         while (t[p].size() <= r-l+1) {
dc 68             int left = get<2>(t[p].back());
ef 4a             if (L > m-1 or (R+m+1 <= r and t[2*p+1][1+R] <
t[2*p][1+L])) {
a3 8c                 t[p].push_back(t[2*p+1][1 + R++]);
8c da                 get<2>(t[p].back()) = left;
62 5e                 continue;
d2 cb             }
62 24             t[p].push_back(t[2*p][1 + L++]);
ec 33             get<2>(t[p].back()) = left+1;
f5 cb         }
bb cb     }

46 dd     int get_l(T y) { return lower_bound(vy.begin(), vy.end(), y) -
vy.begin(); }
62 eb     int get_r(T y) { return upper_bound(vy.begin(), vy.end(), y) -
vy.begin(); }

cc f6     int count(T x1, T x2, T y1, T y2) {
48 90         function<int(int, int, int)> dfs = [&](int p, int l, int
r) {
8b 7c             if (l == r or x2 < get<0>(t[p][0]) or get<1>(t[p][0])
< x1) return 0;
41 2b             if (x1 <= get<0>(t[p][0]) and get<1>(t[p][0]) <= x2)
return r-l;
79 78             int nl = get<2>(t[p][l]), nr = get<2>(t[p][r]);
d0 eb             return dfs(2*p, nl, nr) + dfs(2*p+1, l-nl, r-nr);
20 21         };
68 7c         return dfs(1, get_l(y1), get_r(y2));
5c cb     }
ca 00     vector<int> report(T x1, T x2, T y1, T y2) {
b0 4b         vector<int> ret;
56 85         function<void(int, int, int)> dfs = [&](int p, int l, int
r) {
2c 88             if (l == r or x2 < get<0>(t[p][0]) or get<1>(t[p][0])
< x1) return;
d9 8d             if (x1 <= get<0>(t[p][0]) and get<1>(t[p][0]) <= x2) {
e6 e0                 for (int i = l; i < r; i++)

```



```

    ret.push_back(get<1>(t[p][i+1]));
87 50         return;
24 cb         }
e8 78         int nl = get<2>(t[p][l]), nr = get<2>(t[p][r]);
e2 19         dfs(2*p, nl, nr), dfs(2*p+1, l-nl, r-nr);
05 21     };
3f 8a     dfs(1, get_l(y1), get_r(y2));
0c ed     return ret;
96 cb     }
23 98     int kth(T y1, T y2, int k) {
d5 90         function<int(int, int, int)> dfs = [&](int p, int l, int
r) {
b7 15             if (k >= r-l) {
3b 94                 k -= r-l;
26 da                 return -1;
75 cb             }
09 8d             if (r-l == 1) return get<1>(t[p][l+1]);
4b 78             int nl = get<2>(t[p][l]), nr = get<2>(t[p][r]);
36 07             int left = dfs(2*p, nl, nr);
21 3b             if (left != -1) return left;
ed 04             return dfs(2*p+1, l-nl, r-nr);
32 21         };
49 7c         return dfs(1, get_l(y1), get_r(y2));
a6 cb     }
1c 21 };

```

## 6.8 Min queue - deque

```

// Tudo O(1) amortizado
// c13c57

```

```

1d 1d template<class T> struct minqueue {
7c 2d     deque<pair<T, int>> q;

f9 3f     void push(T x) {
17 56         int ct = 1;
32 95         while (q.size() and x < q.front().first)
dc 75             ct += q.front().second, q.pop_front();
5d 98         q.emplace_front(x, ct);
0a cb     }
ee 42     void pop() {
ae aa         if (q.back().second > 1) q.back().second--;
96 c5         else q.pop_back();
c6 cb     }
e9 ea     T min() { return q.back().first; }
c1 21 };

```

## 6.9 Min queue - stack

```

// Tudo O(1) amortizado
// fe0cad

```

```

55 55 template<class T> struct minstack {
7a 81     stack<pair<T, T>> s;

ef 3f     void push(T x) {
34 12         if (!s.size()) s.push({x, x});
d2 9d         else s.emplace(x, std::min(s.top().second, x));
65 cb     }
1c 4f     T top() { return s.top().first; }
37 94     T pop() {
08 1f         T ans = s.top().first;
29 2e         s.pop();
7b ba         return ans;
5c cb     }
e9 61     int size() { return s.size(); }
59 13     T min() { return s.top().second; }
4c 21 };

b5 1d template<class T> struct minqueue {
02 cd     minstack<T> s1, s2;

04 7c     void push(T x) { s1.push(x); }
5f c9     void move() {
bb d4         if (s2.size()) return;
13 d9         while (s1.size()) {
3a 7a             T x = s1.pop();
23 48             s2.push(x);
3e cb         }
38 cb     }
1d 78     T front() { return move(), s2.top(); }
ae 23     T pop() { return move(), s2.pop(); }
88 7f     int size() { return s1.size()+s2.size(); }
a1 19     T min() {
9b cd         if (!s1.size()) return s2.min();
03 58         else if (!s2.size()) return s1.min();
d5 31         return std::min(s1.min(), s2.min());
3b cb     }
fe 21 };

```

## 6.10 Order Statistic Set

```

// Funciona do C++11 pra cima

```

```
// 901923
77 77 #include <ext/pb_ds/assoc_container.hpp>
07 30 #include <ext/pb_ds/tree_policy.hpp>
99 0d using namespace __gnu_pbds;
e5 4f template <class T>
bd de     using ord_set = tree<T, null_type, less<T>, rb_tree_tag,
90 3a     tree_order_statistics_node_update>;
```

```
// para declarar:
// ord_set<int> s;
// coisas do set normal funcionam:
// for (auto i : s) cout << i << endl;
// cout << s.size() << endl;
// k-esimo maior elemento O(log|s|):
// k=0: menor elemento
// cout << *s.find_by_order(k) << endl;
// quantos sao menores do que k O(log|s|):
// cout << s.order_of_key(k) << endl;
```

```
// Para fazer um multiset, tem que
// usar ord_set<pair<int, int>> com o
// segundo parametro sendo algo para diferenciar
// os elementos iguais.
// s.order_of_key({k, -INF}) vai retornar o
// numero de elementos < k
```

## 6.11 Priority Queue DS

```
// Mantem updates aplicados em uma estrutura de dados
// que permita rollback e nao seja amortizada.
// Cada update possui uma prioridade,
// sendo possivel remover o update com maior prioridade.
// Os updates devem ser comutativos, ou seja, o estado
// da estrutura deve ser o mesmo independente da ordem
// que eles sejam aplicados.
//
// Complexidades:
// update - O(log(n) + T(n))
// query - T(n)
// pop - O(log(n) * T(n)) amortizado
//
// onde T(n) eh a complexidade do update
//
// 54a75e

// assumes all priorities are distinct
94 94 template<typename DS, typename UPD> struct priority_queue_ds {
```

```
cf df     DS D;
6f a7     vector<tuple<UPD, int, int>> upd; // {u, p, idx_in_pos}
9b 86     set<pair<int, int>> st;
d5 92     vector<int> pos;

d9 cf     priority_queue_ds(int n) : D(n) {}

26 6a     void update(UPD u, int p) {
84 9a         D.update(u);
00 d0         st.emplace(p, pos.size());
37 6c         upd.emplace_back(u, p, pos.size());
05 e3         pos.push_back(upd.size() - 1);
60 cb     }

f6 42     int query(int a) {
c7 aa         return D.find(a);
09 cb     }

b3 42     void pop() {
a3 25         int k = 1, min_p; // k = number of pops we will do
1d 43         vector<tuple<UPD, int, int>> small, big;
2a 63         auto it = st.end();
42 23         for (int qt = 0; qt++ < (k+1)/2;) {
c5 04             it--;
4c 3a             min_p = it->first;
35 80             int i = pos[it->second];
9a e8             if (qt > 1) big.push_back(upd[i]);
39 84             k = max<int>(k, upd.size() - i);
19 cb         }

9e b3         for (int i = 0; i < k; i++) {
eb a6             D.rollback();
0f 6d             auto [u, p, idx] = upd.rbegin()[i];
d0 86             if (p < min_p) small.emplace_back(u, p, idx);
36 cb         }

5e 23         st.erase(prev(st.end()));
07 62         upd.erase(upd.end() - k, upd.end());

a9 a2         small.insert(small.end(), big.rbegin(), big.rend());
00 06         for (auto [u, p, idx] : small) {
69 9a             D.update(u);
23 c8             upd.emplace_back(u, p, idx);
ef a7             pos[idx] = upd.size() - 1;
11 cb         }
c7 cb     }
54 21 };
```

## 6.12 Range color

```
// update(l, r, c) colore o range [l, r] com a cor c,
// e retorna os ranges que foram coloridos {l, r, cor}
// query(i) retorna a cor da posicao i
//
// Complexidades (para q operacoes):
// update - O(log(q)) amortizado
// query - O(log(q))
// 9e9cab

df df template<typename T> struct color {
31 f0     set<tuple<int, int, T>> se;

c8 07     vector<tuple<int, int, T>> update(int l, int r, T val) {
8c 9c         auto it = se.upper_bound({r, INF, val});
fd 75         if (it != se.begin() and get<1>(*prev(it)) > r) {
b0 e9             auto [L, R, V] = *--it;
fd 3f             se.erase(it);
98 bf             se.emplace(L, r, V), se.emplace(r+1, R, V);
15 cb         }
1e d9         it = se.lower_bound({l, -INF, val});
42 51         if (it != se.begin() and get<1>(*prev(it)) >= l) {
da e9             auto [L, R, V] = *--it;
c0 3f             se.erase(it);
55 75             se.emplace(L, l-1, V), it = se.emplace(l, R, V).first;
e0 cb         }
9f d7         vector<tuple<int, int, T>> ret;
a7 7a         for (; it != se.end() and get<0>(*it) <= r; it =
se.erase(it))
10 8c             ret.push_back(*it);
f9 b4         se.emplace(l, r, val);
63 ed         return ret;
76 cb     }
09 ff     T query(int i) {
a0 c3         auto it = se.upper_bound({i, INF, T()});
5b 8e         if (it == se.begin() or get<1>(*--it) < i) return -1; //
nao tem
6b 53         return get<2>(*it);
79 cb     }
9e 21 };
```

## 6.13 RMQ <O(n), O(1)> - min queue

```
// O(n) pra buildar, query O(1)
// Se tiver varios minimos, retorna
// o de menor indice
```

```
// bab412
```

```
1a 1a template<typename T> struct rmq {
9e 51     vector<T> v;
4b fc     int n; static const int b = 30;
52 70     vector<int> mask, t;

4e 18     int op(int x, int y) { return v[x] <= v[y] ? x : y; }
a9 ee     int msb(int x) { return __builtin_clz(1)-__builtin_clz(x); }
a1 c9     int small(int r, int sz = b) { return
r-msb(mask[r]&((1<<sz)-1)); }
ed 6a     rmq() {}
e7 43     rmq(const vector<T>& v_) : v(v_), n(v.size()), mask(n), t(n) {
0e 2e         for (int i = 0, at = 0; i < n; mask[i++] = at |= 1) {
4c a6             at = (at<<1)&((1<<b)-1);
87 c0             while (at and op(i-msb(at&-at), i) == i) at ^= at&-at;
47 cb         }
8e ea         for (int i = 0; i < n/b; i++) t[i] = small(b*i+b-1);
5b 39         for (int j = 1; (1<<j) <= n/b; j++) for (int i = 0;
i+(1<<j) <= n/b; i++)
c7 ba             t[n/b*j+i] = op(t[n/b*(j-1)+i],
t[n/b*(j-1)+i+(1<<(j-1))]);
cf cb     }
e2 e3     int index_query(int l, int r) {
dc 27         if (r-l+1 <= b) return small(r, r-l+1);
f8 e8         int x = l/b+1, y = r/b-1;
d6 fd         if (x > y) return op(small(l+b-1), small(r));
97 a4         int j = msb(y-x+1);
b6 ea         int ans = op(small(l+b-1), op(t[n/b*j+x],
t[n/b*j+y-(1<<j)+1]));
1e be         return op(ans, small(r));
d4 cb     }
30 09     T query(int l, int r) { return v[index_query(l, r)]; }
ba 21 };
```

## 6.14 SegTreap

```
// Muda uma posicao do plano, e faz query de operacao
// associativa e comutativa em retangulo
// Mudar ZERO e op
// Esparso nas duas coordenadas, inicialmente eh tudo ZERO
//
// Para query com distancia de manhattan <= d, faca
// nx = x+y, ny = x-y
// Update em (nx, ny), query em ((nx-d, ny-d), (nx+d, ny+d))
//
// Valores no X tem que ser de 0 ateh NX
```

```

// Para q operacoes, usa O(q log(NX)) de memoria, e as
// operacoes custa O(log(q) log(NX))
// 75f2d0

55 55 const int ZERO = INF;
4f 56 const int op(int l, int r) { return min(l, r); }

72 87 mt19937 rng((int)
    chrono::steady_clock::now().time_since_epoch().count());

b1 aa template<typename T> struct treap {
6e 3c     struct node {
7d b1         node *l, *r;
62 ee         int p;
82 85         pair<ll, ll> idx; // {y, x}
e5 36         T val, mi;
6e bc         node(ll x, ll y, T val_) : l(NULL), r(NULL), p(rng()),
64 1b             idx(pair(y, x)), val(val_), mi(val) {}
9b 01         void update() {
95 d6             mi = val;
56 18             if (l) mi = op(mi, l->mi);
f0 b6             if (r) mi = op(mi, r->mi);
6c cb         }
e5 21     };

5d bb     node* root;

35 84     treap() { root = NULL; }
6c ce     ~treap() {
e0 60         vector<node*> q = {root};
df 40         while (q.size()) {
01 e5             node* x = q.back(); q.pop_back();
24 ee             if (!x) continue;
80 1c             q.push_back(x->l), q.push_back(x->r);
7d bf             delete x;
b9 cb         }
f5 cb     }
f4 22     treap(treap&& t) : treap() { swap(root, t.root); }

c3 bc     void join(node* l, node* r, node*& i) { // assume que l < r
c5 98         if (!l or !r) return void(i = l ? l : r);
b0 80         if (l->p > r->p) join(l->r, r, l->r), i = l;
49 fa         else join(l, r->l, r->l), i = r;
a9 bd         i->update();
8a cb     }
ef c8     void split(node* i, node*& l, node*& r, pair<ll, ll> idx) {
8d 26         if (!i) return void(r = l = NULL);

```

```

57 13         if (i->idx < idx) split(i->r, i->r, r, idx), l = i;
5c d2         else split(i->l, l, i->l, idx), r = i;
19 bd         i->update();
53 cb     }
fd d3     void update(ll x, ll y, T v) {
1b df         node *L, *M, *R;
00 8b         split(root, M, R, pair(y, x+1)), split(M, L, M, pair(y,
x));
a0 1e         if (M) M->val = M->mi = v;
a8 9e         else M = new node(x, y, v);
e6 69         join(L, M, M), join(M, R, root);
7e cb     }
bd 91     T query(ll ly, ll ry) {
f7 df         node *L, *M, *R;
9c 1c         split(root, M, R, pair(ry, LINF)), split(M, L, M, pair(ly,
0));
38 0f         T ret = M ? M->mi : ZERO;
2f 69         join(L, M, M), join(M, R, root);
34 ed         return ret;
b3 cb     }
66 21 };

7b 46 template<typename T> struct segtreap {
25 c4     vector<treap<T>> seg;
b5 6e     vector<int> ch[2];
9c e4     ll NX;

60 25     segtreap(ll NX_) : seg(1), NX(NX_) { ch[0].push_back(-1),
ch[1].push_back(-1); }

2b a7     int get_ch(int i, int d){
61 e5         if (ch[d][i] == -1) {
86 2d             ch[d][i] = seg.size();
0c 23             seg.emplace_back();
82 84             ch[0].push_back(-1), ch[1].push_back(-1);
d7 cb         }
66 96         return ch[d][i];
27 cb     }

73 10     T query(ll lx, ll rx, ll ly, ll ry, int p, ll l, ll r) {
d3 00         if (rx < l or r < lx) return ZERO;
77 f0         if (lx <= l and r <= rx) return seg[p].query(ly, ry);

90 e6         ll m = l + (r-l)/2;
44 35         return op(query(lx, rx, ly, ry, get_ch(p, 0), l, m),
3f 06             query(lx, rx, ly, ry, get_ch(p, 1), m+1, r));
2d cb     }

```

```

19 f4    T query(ll lx, ll rx, ll ly, ll ry) { return query(lx, rx, ly,
    ry, 0, 0, NX); }

29 24    void update(ll x, ll y, T val, int p, ll l, ll r) {
44 73        if (l == r) return seg[p].update(x, y, val);
ee e6        ll m = l + (r-l)/2;
dd cc        if (x <= m) update(x, y, val, get_ch(p, 0), l, m);
95 5a        else update(x, y, val, get_ch(p, 1), m+1, r);
40 98        seg[p].update(x, y, val);
9b cb    }
92 51    void update(ll x, ll y, T val) { update(x, y, val, 0, 0, NX); }
75 21 };

```

## 6.15 SegTree

```

// Recursiva com Lazy Propagation
// Query: soma do range [a, b]
// Update: soma x em cada elemento do range [a, b]
// Pode usar a seguinte funcao para indexar os nohs:
// f(l, r) = (l+r)|(l!=r), usando 2N de memoria
//
// Complexidades:
// build - O(n)
// query - O(log(n))
// update - O(log(n))

// 0afec1
aa aa namespace seg {
ae 00    ll seg[4*MAX], lazy[4*MAX];
36 05    int n, *v;

c3 d2    ll build(int p=1, int l=0, int r=n-1) {
c0 3c        lazy[p] = 0;
46 6c        if (l == r) return seg[p] = v[l];
14 ee        int m = (l+r)/2;
33 19        return seg[p] = build(2*p, l, m) + build(2*p+1, m+1, r);
12 cb    }
8f 0d    void build(int n2, int* v2) {
ee 68        n = n2, v = v2;
57 6f        build();
b6 cb    }
ac ce    void prop(int p, int l, int r) {
6d cd        seg[p] += lazy[p]*(r-l+1);
a9 2c        if (l != r) lazy[2*p] += lazy[p], lazy[2*p+1] += lazy[p];
dd 3c        lazy[p] = 0;
d4 cb    }
e9 2c    ll query(int a, int b, int p=1, int l=0, int r=n-1) {

```

```

e2 6b        prop(p, l, r);
1d 52        if (a <= l and r <= b) return seg[p];
be 78        if (b < l or r < a) return 0;
44 ee        int m = (l+r)/2;
31 b1        return query(a, b, 2*p, l, m) + query(a, b, 2*p+1, m+1, r);
43 cb    }
d3 cf    ll update(int a, int b, int x, int p=1, int l=0, int r=n-1) {
c2 6b        prop(p, l, r);
9f 9a        if (a <= l and r <= b) {
dd b9            lazy[p] += x;
70 6b            prop(p, l, r);
f8 53            return seg[p];
a7 cb        }
a6 e9        if (b < l or r < a) return seg[p];
52 ee        int m = (l+r)/2;
5c fd        return seg[p] = update(a, b, x, 2*p, l, m) +
f8 7f            update(a, b, x, 2*p+1, m+1, r);
c9 cb    }
0a 21 };

// Se tiver uma seg de max, da pra descobrir em O(log(n))
// o primeiro e ultimo elemento >= val numa range:

// primeira posicao >= val em [a, b] (ou -1 se nao tem)
// 68c3e5
ed 11 int get_left(int a, int b, int val, int p=1, int l=0, int r=n-1)
    {
5e 6b    prop(p, l, r);
7c f3    if (b < l or r < a or seg[p] < val) return -1;
8e 20    if (r == l) return l;
25 ee    int m = (l+r)/2;
78 75    int x = get_left(a, b, val, 2*p, l, m);
0e 50    if (x != -1) return x;
a6 c3    return get_left(a, b, val, 2*p+1, m+1, r);
6b cb }

// ultima posicao >= val em [a, b] (ou -1 se nao tem)
// 1b71df
d2 99 int get_right(int a, int b, int val, int p=1, int l=0, int
    r=n-1) {
19 6b    prop(p, l, r);
f4 f3    if (b < l or r < a or seg[p] < val) return -1;
23 20    if (r == l) return l;
b5 ee    int m = (l+r)/2;
2a 1b    int x = get_right(a, b, val, 2*p+1, m+1, r);
26 50    if (x != -1) return x;

```

```

52 6a    return get_right(a, b, val, 2*p, l, m);
0d cb }

// Se tiver uma seg de soma sobre um array nao negativo v, da pra
// descobrir em O(log(n)) o maior j tal que v[i]+v[i+1]+...+v[j-1] <
// val
// 2b8ea7
5e 6a int lower_bound(int i, ll& val, int p, int l, int r) {
ed 6b     prop(p, l, r);
39 6e     if (r < i) return n;
5d b5     if (i <= l and seg[p] < val) {
67 bf         val -= seg[p];
31 04         return n;
97 cb     }
5c 3c     if (l == r) return l;
ba ee     int m = (l+r)/2;
18 51     int x = lower_bound(i, val, 2*p, l, m);
fb ee     if (x != n) return x;
b9 8b     return lower_bound(i, val, 2*p+1, m+1, r);
30 cb }

```

## 6.16 SegTree 2D Iterativa

```

// Consultas 0-based
// Um valor inicial em (x, y) deve ser colocado em seg[x+n][y+n]
// Query: soma do retangulo ((x1, y1), (x2, y2))
// Update: muda o valor da posicao (x, y) para val
// Nao pergunte como que essa coisa funciona
//
// Para query com distancia de manhattan <= d, faca
// nx = x+y, ny = x-y
// Update em (nx, ny), query em ((nx-d, ny-d), (nx+d, ny+d))
//
// Se for de min/max, pode tirar os if's da 'query', e fazer
// sempre as 4 operacoes. Fica mais rapido
//
// Complexidades:
// build - O(n^2)
// query - O(log^2(n))
// update - O(log^2(n))
// 67b9e5

```

```

73 73 int seg[2*MAX][2*MAX], n;

e7 0a void build() {
13 91     for (int x = 2*n; x; x--) for (int y = 2*n; y; y--) {
89 c8         if (x < n) seg[x][y] = seg[2*x][y] + seg[2*x+1][y];

```

```

79 fe         if (y < n) seg[x][y] = seg[x][2*y] + seg[x][2*y+1];
a4 cb     }
63 cb }

fd 25 int query(int x1, int y1, int x2, int y2) {
92 82     int ret = 0, y3 = y1 + n, y4 = y2 + n;
38 83     for (x1 += n, x2 += n; x1 <= x2; ++x1 /= 2, --x2 /= 2)
12 0f         for (y1 = y3, y2 = y4; y1 <= y2; ++y1 /= 2, --y2 /= 2) {
0c 55             if (x1%2 == 1 and y1%2 == 1) ret += seg[x1][y1];
2c 6b             if (x1%2 == 1 and y2%2 == 0) ret += seg[x1][y2];
4d c0             if (x2%2 == 0 and y1%2 == 1) ret += seg[x2][y1];
38 5d             if (x2%2 == 0 and y2%2 == 0) ret += seg[x2][y2];
7e cb         }

54 ed     return ret;
40 cb }

72 76 void update(int x, int y, int val) {
cb 66     int y2 = y += n;
69 19     for (x += n; x; x /= 2, y = y2) {
86 97         if (x >= n) seg[x][y] = val;
8a ba         else seg[x][y] = seg[2*x][y] + seg[2*x+1][y];

b5 3b         while (y /= 2) seg[x][y] = seg[x][2*y] + seg[x][2*y+1];
fa cb     }
67 cb }

```

## 6.17 SegTree Beats

```

// query(a, b) - {{min(v[a..b]), max(v[a..b])}, sum(v[a..b])}
// updatemin(a, b, x) faz com que v[i] <- min(v[i], x),
// para i em [a, b]
// updatemax faz o mesmo com max, e updatesum soma x
// em todo mundo do intervalo [a, b]
//
// Complexidades:
// build - O(n)
// query - O(log(n))
// update - O(log^2(n)) amortizado
// (se nao usar updatesum, fica log(n) amortizado)
// 41672b

```

```

7c 7c #define f first
8b 0a #define s second

```

```

95 f3 namespace beats {
f3 3c     struct node {

```

```

95 52      int tam;
c9 12      ll sum, lazy; // lazy pra soma
ed 4f      ll mi1, mi2, mi; // mi = #mi1
4f c6      ll ma1, ma2, ma; // ma = #ma1

ee 42      node(ll x = 0) {
dd ba          sum = mi1 = ma1 = x;
1f b2          mi2 = LINF, ma2 = -LINF;
0c 62          mi = ma = tam = 1;
e3 c6          lazy = 0;
ea cb      }
a5 77      node(const node& l, const node& r) {
83 a9          sum = l.sum + r.sum, tam = l.tam + r.tam;
7d c6          lazy = 0;
be 79          if (l.mi1 > r.mi1) {
a0 23              mi1 = r.mi1, mi = r.mi;
7b ea              mi2 = min(l.mi1, r.mi2);
57 dc          } else if (l.mi1 < r.mi1) {
b9 e3              mi1 = l.mi1, mi = l.mi;
ac 4b              mi2 = min(r.mi1, l.mi2);
e1 9d          } else {
1f a3              mi1 = l.mi1, mi = l.mi+r.mi;
90 83              mi2 = min(l.mi2, r.mi2);
08 cb          }
cb cd          if (l.ma1 < r.ma1) {
75 6a              ma1 = r.ma1, ma = r.ma;
67 96              ma2 = max(l.ma1, r.ma2);
60 5f          } else if (l.ma1 > r.ma1) {
68 ae              ma1 = l.ma1, ma = l.ma;
b8 2c              ma2 = max(r.ma1, l.ma2);
75 9d          } else {
0c db              ma1 = l.ma1, ma = l.ma+r.ma;
43 c0              ma2 = max(l.ma2, r.ma2);
f3 cb          }
77 cb      }
ac 4b      void setmin(ll x) {
e0 55          if (x >= ma1) return;
6b 46          sum += (x - ma1)*ma;
1c be          if (mi1 == ma1) mi1 = x;
78 0a          if (mi2 == ma1) mi2 = x;
53 b8          ma1 = x;
19 cb      }
80 6c      void setmax(ll x) {
75 e2          if (x <= mi1) return;
22 7e          sum += (x - mi1)*mi;
19 0b          if (ma1 == mi1) ma1 = x;
86 c3          if (ma2 == mi1) ma2 = x;

```

```

2f 1f          mi1 = x;
96 cb          }
e1 4c          void setsum(ll x) {
86 fe              mi1 += x, mi2 += x, ma1 += x, ma2 += x;
6d 62              sum += x*tam;
5a c4              lazy += x;
7e cb          }
38 21      };

9a 62      node seg[4*MAX];
f0 05      int n, *v;

2c 93      node build(int p=1, int l=0, int r=n-1) {
94 d8          if (l == r) return seg[p] = {v[l]};
e7 ee          int m = (l+r)/2;
87 3d          return seg[p] = {build(2*p, l, m), build(2*p+1, m+1, r)};
36 cb      }
d0 0d      void build(int n2, int* v2) {
d5 68          n = n2, v = v2;
67 6f          build();
6e cb      }
ba ce      void prop(int p, int l, int r) {
e8 8c          if (l == r) return;
d0 ab          for (int k = 0; k < 2; k++) {
f2 d0              if (seg[p].lazy) seg[2*p+k].setsum(seg[p].lazy);
43 84              seg[2*p+k].setmin(seg[p].ma1);
f1 f7              seg[2*p+k].setmax(seg[p].mi1);
bb cb          }
b1 43          seg[p].lazy = 0;
c7 cb      }
9b 05      pair<pair<ll, ll>, ll> query(int a, int b, int p=1, int l=0,
int r=n-1) {
b9 e0          if (b < l or r < a) return {{LINF, -LINF}, 0};
9a 9b          if (a <= l and r <= b) return {{seg[p].mi1, seg[p].ma1},
seg[p].sum};
ee 6b          prop(p, l, r);
7d ee          int m = (l+r)/2;
46 e6          auto L = query(a, b, 2*p, l, m), R = query(a, b, 2*p+1,
m+1, r);
c4 96          return {{min(L.f.f, R.f.f), max(L.f.s, R.f.s)}, L.s+R.s};
cf cb      }
11 2c      node updatemin(int a, int b, ll x, int p=1, int l=0, int
r=n-1) {
a3 74          if (b < l or r < a or seg[p].ma1 <= x) return seg[p];
5c 30          if (a <= l and r <= b and seg[p].ma2 < x) {
87 cc              seg[p].setmin(x);
c1 53              return seg[p];

```

```

58 cb      }
c0 6b      prop(p, l, r);
51 ee      int m = (l+r)/2;
5a 96      return seg[p] = {updatemin(a, b, x, 2*p, l, m),
89 fa              updatemin(a, b, x, 2*p+1, m+1, r)};
a8 cb      }
42 04      node updatemax(int a, int b, ll x, int p=1, int l=0, int
r=n-1) {
d6 b5      if (b < l or r < a or seg[p].mi1 >= x) return seg[p];
83 a9      if (a <= l and r <= b and seg[p].mi2 > x) {
e2 e8          seg[p].setmax(x);
3a 53          return seg[p];
89 cb      }
4f 6b      prop(p, l, r);
22 ee      int m = (l+r)/2;
37 ee      return seg[p] = {updatemax(a, b, x, 2*p, l, m),
0e bd          updatemax(a, b, x, 2*p+1, m+1, r)};
bc cb      }
01 ae      node updatesum(int a, int b, ll x, int p=1, int l=0, int
r=n-1) {
a3 e9      if (b < l or r < a) return seg[p];
ea 9a      if (a <= l and r <= b) {
65 8f          seg[p].setsum(x);
2b 53          return seg[p];
4e cb      }
4b 6b      prop(p, l, r);
b7 ee      int m = (l+r)/2;
7d 7b      return seg[p] = {updatesum(a, b, x, 2*p, l, m),
bf dd          updatesum(a, b, x, 2*p+1, m+1, r)};
44 cb      }
41 21 };

```

## 6.18 SegTree Colorida

```

// Cada posicao tem um valor e uma cor
// 0 construtor recebe um vector de {valor, cor}
// e o numero de cores (as cores devem estar em [0, c-1])
// query(c, a, b) retorna a soma dos valores
// de todo mundo em [a, b] que tem cor c
// update(c, a, b, x) soma x em todo mundo em
// [a, b] que tem cor c
// paint(c1, c2, a, b) faz com que todo mundo
// em [a, b] que tem cor c1 passe a ter cor c2
//
// Complexidades:
// construir - O(n log(n)) espaco e tempo
// query - O(log(n))

```

```

// update - O(log(n))
// paint - O(log(n)) amortizado
// 2938e8

04 04 struct seg_color {
06 3c     struct node {
8e b1         node *l, *r;
08 0f         int cnt;
23 9c         ll val, lazy;
00 27         node() : l(NULL), r(NULL), cnt(0), val(0), lazy(0) {}
1c 01         void update() {
8f d0             cnt = 0, val = 0;
c3 bc             for (auto i : {l, r}) if (i) {
90 c8                 i->prop();
5c 28                 cnt += i->cnt, val += i->val;
b2 cb             }
c3 cb         }
e1 a9         void prop() {
10 2d             if (!lazy) return;
dc 3f             val += lazy*(ll)cnt;
05 b6             for (auto i : {l, r}) if (i) i->lazy += lazy;
0f c6             lazy = 0;
31 cb         }
43 21     };

73 1a     int n;
95 9b     vector<node*> seg;

66 6e     seg_color(vector<pair<int, int>>& v, int c) : n(v.size()),
seg(c, NULL) {
a9 83         for (int i = 0; i < n; i++)
f2 9b             seg[v[i].second] = insert(seg[v[i].second], i,
v[i].first, 0, n-1);
1e cb     }
b6 3c     ~seg_color() {
2e dd         queue<node*> q;
05 3a         for (auto i : seg) q.push(i);
aa 40         while (q.size()) {
68 20             auto i = q.front(); q.pop();
e3 da             if (!i) continue;
11 7c             q.push(i->l), q.push(i->r);
67 5c             delete i;
3a cb         }
c1 cb     }

ba 40     node* insert(node* at, int idx, int val, int l, int r) {
92 1a         if (!at) at = new node();

```



```

0e 23     if (l == r) return at->cnt = 1, at->val = val, at;
d7 ee     int m = (l+r)/2;
4d 13     if (idx <= m) at->l = insert(at->l, idx, val, l, m);
2c 3e     else at->r = insert(at->r, idx, val, m+1, r);
f5 cf     return at->update(), at;
a8 cb }
27 87 ll query(node* at, int a, int b, int l, int r) {
12 61     if (!at or b < l or r < a) return 0;
d9 d9     at->prop();
ca cb     if (a <= l and r <= b) return at->val;
de ee     int m = (l+r)/2;
4d 4c     return query(at->l, a, b, l, m) + query(at->r, a, b, m+1,
r);
96 cb }
e0 e5 ll query(int c, int a, int b) { return query(seg[c], a, b, 0,
n-1); }
30 91 void update(node* at, int a, int b, int x, int l, int r) {
d6 fb     if (!at or b < l or r < a) return;
32 d9     at->prop();
21 9a     if (a <= l and r <= b) {
97 e9         at->lazy += x;
4b cb         return void(at->prop());
1b cb     }
5f ee     int m = (l+r)/2;
87 0b     update(at->l, a, b, x, l, m), update(at->r, a, b, x, m+1,
r);
73 7b     at->update();
46 cb }
6f a4 void update(int c, int a, int b, int x) { update(seg[c], a, b,
x, 0, n-1); }
84 70 void paint(node*& from, node*& to, int a, int b, int l, int r)
{
11 10     if (to == from or !from or b < l or r < a) return;
0b e8     from->prop();
55 88     if (to) to->prop();
8c 9a     if (a <= l and r <= b) {
20 24         if (!to) {
1f 38             to = from;
1c 14             from = NULL;
9d 50             return;
f9 cb         }
96 ee         int m = (l+r)/2;
62 1c         paint(from->l, to->l, a, b, l, m), paint(from->r,
to->r, a, b, m+1, r);
16 72         to->update();
90 27         delete from;
10 14         from = NULL;

```

```

4b 50         return;
93 cb     }
05 01     if (!to) to = new node();
5c ee     int m = (l+r)/2;
ca 1c     paint(from->l, to->l, a, b, l, m), paint(from->r, to->r,
a, b, m+1, r);
dc 45     from->update(), to->update();
df cb }
cf 47 void paint(int c1, int c2, int a, int b) { paint(seg[c1],
seg[c2], a, b, 0, n-1); }
29 21 };

```

## 6.19 SegTree Esparsa - Lazy

```

// Query: soma do range [a, b]
// Update: flipa os valores de [a, b]
// 0 MAX tem q ser Q log N para Q updates
//
// Complexidades:
// build - O(1)
// query - O(log(n))
// update - O(log(n))
// dc37e6

```

```

aa aa namespace seg {
20 6d     int seg[MAX], lazy[MAX], R[MAX], L[MAX], ptr;
9c e9     int get_l(int i){
e4 3d         if (L[i] == 0) L[i] = ptr++;
b0 a9         return L[i];
69 cb     }
78 94     int get_r(int i){
5c 71         if (R[i] == 0) R[i] = ptr++;
a9 28         return R[i];
59 cb     }

ce e7     void build() { ptr = 2; }

7d ce     void prop(int p, int l, int r) {
08 b7         if (!lazy[p]) return;
db 76         seg[p] = r-l+1 - seg[p];
f5 21         if (l != r) lazy[get_l(p)]^=lazy[p],
lazy[get_r(p)]^=lazy[p];
56 3c         lazy[p] = 0;
e8 cb     }

aa 15     int query(int a, int b, int p=1, int l=0, int r=N-1) {
8e 6b         prop(p, l, r);

```

```

6c 78      if (b < l or r < a) return 0;
ce 52      if (a <= l and r <= b) return seg[p];

a3 ee      int m = (l+r)/2;
7a 81      return query(a, b, get_l(p), l, m)+query(a, b, get_r(p),
m+1, r);
73 cb      }

63 51      int update(int a, int b, int p=1, int l=0, int r=N-1) {
91 6b          prop(p, l, r);
aa e9          if (b < l or r < a) return seg[p];
98 9a          if (a <= l and r <= b) {
b6 ab              lazy[p] ^= 1;
17 6b              prop(p, l, r);
c9 53              return seg[p];
59 cb          }
a9 ee          int m = (l+r)/2;
e2 43          return seg[p] = update(a, b, get_l(p), l, m)+update(a, b,
get_r(p), m+1, r);
b8 cb      }
dc 21      };

```

## 6.20 SegTree Esparsa - O(q) memoria

```

// Query: min do range [a, b]
// Update: troca o valor de uma posicao
// Usa O(q) de memoria para q updates
//
// Complexidades:
// query - O(log(n))
// update - O(log(n))
// 072a21

```

```

13 13      template<typename T> struct seg {
ee 3c          struct node {
7d d5              node* ch[2];
b7 97              char d;
fa ca              T v;

67 c4              T mi;

53 d4              node(int d_, T v_, T val) : d(d_), v(v_) {
53 e7                  ch[0] = ch[1] = NULL;
14 d6                  mi = val;
a7 cb              }
0e b3              node(node* x) : d(x->d), v(x->v), mi(x->mi) {
f5 c9                  ch[0] = x->ch[0], ch[1] = x->ch[1];

```

```

75 cb      }
b4 01      void update() {
3d 90          mi = numeric_limits<T>::max();
d2 15          for (int i = 0; i < 2; i++) if (ch[i])
30 b5              mi = min(mi, ch[i]->mi);
c6 cb      }
d1 21      };

29 bb      node* root;
f3 9c      char n;

c0 ba      seg() : root(NULL), n(0) {}
4c 51      ~seg() {
30 4c          std::vector<node*> q = {root};
87 40          while (q.size()) {
90 e5              node* x = q.back(); q.pop_back();
a2 ee              if (!x) continue;
b0 73              q.push_back(x->ch[0]), q.push_back(x->ch[1]);
07 bf              delete x;
60 cb          }
ff cb      }

36 1a      char msb(T v, char l, char r) { // msb in range (l, r]
7c 8e          for (char i = r; i > l; i--) if (v>>i&1) return i;
2c da          return -1;
2f cb      }
a6 43      void cut(node* at, T v, char i) {
de 67          char d = msb(v ^ at->v, at->d, i);
1e 23          if (d == -1) return; // no need to split
40 eb          node* nxt = new node(at);
fb d4          at->ch[v>>d&1] = NULL;
bf 34          at->ch[!(v>>d&1)] = nxt;
35 15          at->d = d;
f1 cb      }

8b 6e      node* update(node* at, T idx, T val, char i) {
26 c8          if (!at) return new node(-1, idx, val);
26 d6          cut(at, idx, i);
bb 1a          if (at->d == -1) { // leaf
5d 79              at->mi = val;
01 ce              return at;
08 cb          }
13 b2          bool dir = idx>>at->d&1;
9d c8          at->ch[dir] = update(at->ch[dir], idx, val, at->d-1);
45 7b          at->update();
fb ce          return at;
ad cb      }

```

```

b8 85 void update(T idx, T val) {
52 8f while (idx>>n) n++;
50 61 root = update(root, idx, val, n-1);
16 cb }

48 9d T query(node* at, T a, T b, T l, T r, char i) {
12 df if (!at or b < l or r < a) return numeric_limits<T>::max();
47 fd if (a <= l and r <= b) return at->mi;
a3 84 T m = l + (r-l)/2;
22 c8 if (at->d < i) {
b6 c5 if ((at->v>>i&1) == 0) return query(at, a, b, l, m,
i-1);
22 ca else return query(at, a, b, m+1, r, i-1);
3b cb }
b9 37 return min(query(at->ch[0], a, b, l, m, i-1),
query(at->ch[1], a, b, m+1, r, i-1));
81 cb }
2f 6f T query(T l, T r) { return query(root, l, r, 0, (1<<n)-1,
n-1); }
07 21 };

```

## 6.21 SegTree Iterativa

```

// Consultas 0-based
// Valores iniciais devem estar em (seg[n], ... , seg[2*n-1])
// Query: soma do range [a, b]
// Update: muda o valor da posicao p para x
//
// Complexidades:
// build - O(n)
// query - O(log(n))
// update - O(log(n))
// 779519

6a 6a int seg[2 * MAX];
ce 1a int n;

fa 0a void build() {
c9 d1 for (int i = n - 1; i; i--) seg[i] = seg[2*i] + seg[2*i+1];
d3 cb }

e1 4e int query(int a, int b) {
12 7c int ret = 0;
de 72 for(a += n, b += n; a <= b; ++a /= 2, --b /= 2) {
8f 4e if (a % 2 == 1) ret += seg[a];
52 24 if (b % 2 == 0) ret += seg[b];
b5 cb }

```

```

91 ed return ret;
ae cb }

ad ff void update(int p, int x) {
d6 37 seg[p += n] = x;
80 c8 while (p /= 2) seg[p] = seg[2*p] + seg[2*p+1];
77 cb }

```

## 6.22 SegTree Iterativa com Lazy Propagation

```

// Query: soma do range [a, b]
// Update: soma x em cada elemento do range [a, b]
// Para mudar, mudar as funcoes junta, poe e query
// LOG = ceil(log2(MAX))
//
// Complexidades:
// build - O(n)
// query - O(log(n))
// update - O(log(n))
// 6dc475

aa aa namespace seg {
1a 6d ll seg[2*MAX], lazy[2*MAX];
0e 1a int n;

d1 9b ll junta(ll a, ll b) {
3f 53 return a+b;
88 cb }

// soma x na posicao p de tamanho tam
b3 1b void poe(int p, ll x, int tam, bool prop=1) {
ba 51 seg[p] += x*tam;
94 6a if (prop and p < n) lazy[p] += x;
5e cb }

// atualiza todos os pais da folha p
46 b1 void sobe(int p) {
8f d5 for (int tam = 2; p /= 2; tam *= 2) {
27 4c seg[p] = junta(seg[2*p], seg[2*p+1]);
1e 38 poe(p, lazy[p], tam, 0);
b6 cb }
8f cb }

// propaga o caminho da raiz ate a folha p
b9 a0 void prop(int p) {
f0 07 int tam = 1 << (LOG-1);
d8 0a for (int s = LOG; s; s--, tam /= 2) {

```

```

33 4b         int i = p >> s;
e0 27         if (lazy[i]) {
ff 86             poe(2*i, lazy[i], tam);
ab e3             poe(2*i+1, lazy[i], tam);
c7 b9             lazy[i] = 0;
ca cb         }
95 cb     }
f5 cb }

88 61 void build(int n2, int* v) {
11 1e     n = n2;
1b 95     for (int i = 0; i < n; i++) seg[n+i] = v[i];
72 c4     for (int i = n-1; i; i--) seg[i] = junta(seg[2*i],
seg[2*i+1]);
e9 f4     for (int i = 0; i < 2*n; i++) lazy[i] = 0;
71 cb }

60 4f ll query(int a, int b) {
60 b7     ll ret = 0;
61 b4     for (prop(a+=n), prop(b+=n); a <= b; ++a/=2, --b/=2) {
1d a8         if (a%2 == 1) ret = junta(ret, seg[a]);
ce c5         if (b%2 == 0) ret = junta(ret, seg[b]);
9a cb     }
e5 ed     return ret;
dc cb }

b5 a2 void update(int a, int b, int x) {
2e c2     int a2 = a += n, b2 = b += n, tam = 1;
47 0f     for (; a <= b; ++a/=2, --b/=2, tam *= 2) {
42 32         if (a%2 == 1) poe(a, x, tam);
f1 9d         if (b%2 == 0) poe(b, x, tam);
27 cb     }
34 0f     sobe(a2), sobe(b2);
81 cb }
6d 21 };

```

## 6.23 SegTree PA

```

// Segtree de PA
// update_set(l, r, A, R) seta [l, r] para PA(A, R),
// update_add soma PA(A, R) em [l, r]
// query(l, r) retorna a soma de [l, r]
//
// PA(A, R) eh a PA: [A+R, A+2R, A+3R, ... ]
//
// Complexidades:
// construir - O(n)

```

```

// update_set, update_add, query - O(log(n))
// bc4746

dc dc struct seg_pa {
08 35     struct Data {
b3 8f         ll sum;
4b 66         ll set_a, set_r, add_a, add_r;
bd 9b         Data() : sum(0), set_a(LINF), set_r(0), add_a(0), add_r(0)
            {}
4b 21     };
14 16     vector<Data> seg;
64 1a     int n;

92 d4     seg_pa(int n_) {
87 e9         n = n_;
70 fc         seg = vector<Data>(4*n);
bd cb     }

f7 ce     void prop(int p, int l, int r) {
39 d5         int tam = r-l+1;
09 c3         ll &sum = seg[p].sum, &set_a = seg[p].set_a, &set_r =
seg[p].set_r,
29 a1         &add_a = seg[p].add_a, &add_r = seg[p].add_r;

f2 c0         if (set_a != LINF) {
e8 66             set_a += add_a, set_r += add_r;
46 06             sum = set_a*tam + set_r*tam*(tam+1)/2;
ea 57             if (l != r) {
7e ee                 int m = (l+r)/2;

f3 88                 seg[2*p].set_a = set_a;
11 35                 seg[2*p].set_r = set_r;
78 ed                 seg[2*p].add_a = seg[2*p].add_r = 0;

c7 f0                 seg[2*p+1].set_a = set_a + set_r * (m-l+1);
bb 47                 seg[2*p+1].set_r = set_r;
72 d4                 seg[2*p+1].add_a = seg[2*p+1].add_r = 0;
4f cb             }
bd 82             set_a = LINF, set_r = 0;
8a 95             add_a = add_r = 0;
0e 10         } else if (add_a or add_r) {
39 18             sum += add_a*tam + add_r*tam*(tam+1)/2;
34 57             if (l != r) {
67 ee                 int m = (l+r)/2;

22 ff                 seg[2*p].add_a += add_a;
63 ec                 seg[2*p].add_r += add_r;

```

```

7e 06         seg[2*p+1].add_a += add_a + add_r * (m-l+1);
65 a6         seg[2*p+1].add_r += add_r;
d3 cb     }
02 95     add_a = add_r = 0;
1d cb     }
d6 cb }

22 0b int inter(pair<int, int> a, pair<int, int> b) {
78 98     if (a.first > b.first) swap(a, b);
6c ee     return max(0, min(a.second, b.second) - b.first + 1);
ac cb }

bf be ll set(int a, int b, ll aa, ll rr, int p, int l, int r) {
c2 6b     prop(p, l, r);
37 45     if (b < l or r < a) return seg[p].sum;
65 9a     if (a <= l and r <= b) {
0d 91         seg[p].set_a = aa;
eb 77         seg[p].set_r = rr;
31 6b         prop(p, l, r);
84 25         return seg[p].sum;
e6 cb     }
8e ee     int m = (l+r)/2;
28 96     int tam_l = inter({l, m}, {a, b});
75 c3     return seg[p].sum = set(a, b, aa, rr, 2*p, l, m) +
1f 36         set(a, b, aa + rr * tam_l, rr, 2*p+1, m+1, r);
90 cb }

bf f5 void update_set(int l, int r, ll aa, ll rr) {
e6 6f     set(l, r, aa, rr, 1, 0, n-1);
bb cb }

b2 5f ll add(int a, int b, ll aa, ll rr, int p, int l, int r) {
b1 6b     prop(p, l, r);
9e 45     if (b < l or r < a) return seg[p].sum;
89 9a     if (a <= l and r <= b) {
11 35         seg[p].add_a += aa;
96 1e         seg[p].add_r += rr;
d9 6b         prop(p, l, r);
b7 25         return seg[p].sum;
28 cb     }
b4 ee     int m = (l+r)/2;
9c 96     int tam_l = inter({l, m}, {a, b});
c1 58     return seg[p].sum = add(a, b, aa, rr, 2*p, l, m) +
10 69         add(a, b, aa + rr * tam_l, rr, 2*p+1, m+1, r);
88 cb }

85 84 void update_add(int l, int r, ll aa, ll rr) {
1b af     add(l, r, aa, rr, 1, 0, n-1);
7e cb }

2c f4 ll query(int a, int b, int p, int l, int r) {

```

```

f1 6b         prop(p, l, r);
23 78         if (b < l or r < a) return 0;
68 e9         if (a <= l and r <= b) return seg[p].sum;
fe ee         int m = (l+r)/2;
40 b1         return query(a, b, 2*p, l, m) + query(a, b, 2*p+1, m+1, r);
8a cb     }
42 bf     ll query(int l, int r) { return query(l, r, 1, 0, n-1); }
bc 21 };

```

## 6.24 SegTree Persistente

```

// SegTree de soma, update de somar numa posicao
//
// query(a, b, t) retorna a query de [a, b] na versao t
// update(a, x, t) faz um update v[a]+=x a partir da
// versao de t, criando uma nova versao e retornando seu id
// Por default, faz o update a partir da ultima versao
//
// build - O(n)
// query - O(log(n))
// update - O(log(n))
// 50ab73

54 54 const int MAX = 1e5+10, UPD = 1e5+10, LOG = 18;
59 6d const int MAXS = 2*MAX+UPD*LOG;

53 f6 namespace perseg {
9a bd     ll seg[MAXS];
43 f4     int rt[UPD], L[MAXS], R[MAXS], cnt, t;
79 05     int n, *v;

fe 3c     ll build(int p, int l, int r) {
b9 6c         if (l == r) return seg[p] = v[l];
32 85         L[p] = cnt++, R[p] = cnt++;
33 ee         int m = (l+r)/2;
97 27         return seg[p] = build(L[p], l, m) + build(R[p], m+1, r);
ef cb     }
79 0d     void build(int n2, int* v2) {
97 68         n = n2, v = v2;
c7 85         rt[0] = cnt++;
e6 c5         build(0, 0, n-1);
ed cb     }

2a f4     ll query(int a, int b, int p, int l, int r) {
13 78         if (b < l or r < a) return 0;
b2 52         if (a <= l and r <= b) return seg[p];
54 ee         int m = (l+r)/2;
64 1e         return query(a, b, L[p], l, m) + query(a, b, R[p], m+1, r);

```

```

2c cb    }
3f 18    ll query(int a, int b, int tt) {
6f c1        return query(a, b, rt[tt], 0, n-1);
42 cb    }
2e bb    ll update(int a, int x, int lp, int p, int l, int r) {
45 74        if (l == r) return seg[p] = seg[lp]+x;
c8 ee        int m = (l+r)/2;
72 ab        if (a <= m)
6b b4            return seg[p] = update(a, x, L[lp], L[p]=cnt++, l, m)
+ seg[R[p]=R[lp]];
06 8a        return seg[p] = seg[L[p]=L[lp]] + update(a, x, R[lp],
R[p]=cnt++, m+1, r);
d6 cb    }
68 6f    int update(int a, int x, int tt=t) {
3e ab        update(a, x, rt[tt], rt[tt]=cnt++, 0, n-1);
84 e0        return t;
50 cb    }
50 21 };

```

## 6.25 Sparse Table

```

// Resolve RMQ
// MAX2 = log(MAX)
//
// Complexidades:
// build - O(n log(n))
// query - O(1)
// 7aa4c9

```

```

cc cc namespace sparse {
ca 71    int m[MAX2][MAX], n;
11 61    void build(int n2, int* v) {
df 1e        n = n2;
fb 78        for (int i = 0; i < n; i++) m[0][i] = v[i];
e3 a1        for (int j = 1; (1<<j) <= n; j++) for (int i = 0; i+(1<<j)
<= n; i++)
cc 5d            m[j][i] = min(m[j-1][i], m[j-1][i+(1<<(j-1))]);
88 cb    }
8d 4e    int query(int a, int b) {
05 ee        int j = __builtin_clz(1) - __builtin_clz(b-a+1);
66 dc        return min(m[j][a], m[j][b-(1<<j)+1]);
7a cb    }
7a cb }

```

## 6.26 Sparse Table Disjunta

```

// Resolve qualquer operacao associativa

```

```

// MAX2 = log(MAX)
//
// Complexidades:
// build - O(n log(n))
// query - O(1)
// fd81ae

```

```

cc cc namespace sparse {
90 9b    int m[MAX2][2*MAX], n, v[2*MAX];
ba 5f    int op(int a, int b) { return min(a, b); }
27 0d    void build(int n2, int* v2) {
1f 1e        n = n2;
45 df        for (int i = 0; i < n; i++) v[i] = v2[i];
d7 a8        while (n&(n-1)) n++;
21 3d        for (int j = 0; (1<<j) < n; j++) {
d3 1c            int len = 1<<j;
c3 d9            for (int c = len; c < n; c += 2*len) {
58 33                m[j][c] = v[c], m[j][c-1] = v[c-1];
a0 66                for (int i = c+1; i < c+len; i++) m[j][i] =
op(m[j][i-1], v[i]);
ef 43                for (int i = c-2; i >= c-len; i--) m[j][i] =
op(v[i], m[j][i+1]);
fd cb            }
dd cb        }
41 cb    }
ae 9e    int query(int l, int r) {
5f f1        if (l == r) return v[l];
22 e6        int j = __builtin_clz(1) - __builtin_clz(l~r);
a1 d6        return op(m[j][l], m[j][r]);
55 cb    }
fd cb }

```

## 6.27 Splay Tree

```

// SEMPRE QUE DESCER NA ARVORE, DAR SPLAY NO
// NODE MAIS PROFUNDO VISITADO
// Todas as operacoes sao O(log(n)) amortizado
// Se quiser colocar mais informacao no node,
// mudar em 'update'
// 4ff2b3

```

```

53 53 template<typename T> struct splaytree {
ff 3c    struct node {
29 18        node *ch[2], *p;
bc e4        int sz;
5e f4        T val;
d9 da        node(T v) {

```

```

6f 69         ch[0] = ch[1] = p = NULL;
35 a2         sz = 1;
7a 25         val = v;
0a cb     }
9c 01     void update() {
0a a2         sz = 1;
db c7         for (int i = 0; i < 2; i++) if (ch[i]) {
2b d5             sz += ch[i]->sz;
7a cb         }
1e cb     }
db 21 };

cc bb     node* root;

9f fb     splaytree() { root = NULL; }
ff 21     splaytree(const splaytree& t) {
d2 cb         throw logic_error("Nao copiar a splaytree!");
75 cb     }
54 89     ~splaytree() {
a0 60         vector<node*> q = {root};
0e 40         while (q.size()) {
48 e5             node* x = q.back(); q.pop_back();
8f ee             if (!x) continue;
19 73             q.push_back(x->ch[0]), q.push_back(x->ch[1]);
16 bf             delete x;
60 cb         }
ad cb     }

88 94     void rotate(node* x) { // x vai ficar em cima
5c d9         node *p = x->p, *pp = p->p;
98 ec         if (pp) pp->ch[pp->ch[1] == p] = x;
5f 28         bool d = p->ch[0] == x;
0f d6         p->ch[!d] = x->ch[d], x->ch[d] = p;
08 ba         if (p->ch[!d]) p->ch[!d]->p = p;
1c fc         x->p = pp, p->p = x;
a7 1e         p->update(), x->update();
ad cb     }

c4 3f     node* splay(node* x) {
5b a3         if (!x) return x;
93 4e         root = x;
58 3c         while (x->p) {
b6 d9             node *p = x->p, *pp = p->p;
30 35             if (!pp) return rotate(x), x; // zig
12 e3             if ((pp->ch[0] == p)^(p->ch[0] == x))
18 a2                 rotate(x), rotate(x); // zigzag
91 4b             else rotate(p), rotate(x); // zigzig
d9 cb         }

```

```

06 ea         return x;
ba cb     }
53 31     node* insert(T v, bool lb=0) {
e9 b6         if (!root) return lb ? NULL : root = new node(v);
c7 00         node *x = root, *last = NULL;;
03 31         while (1) {
8d 5d             bool d = x->val < v;
81 0f             if (!d) last = x;
2c c2             if (x->val == v) break;
c4 c1             if (x->ch[d]) x = x->ch[d];
7b 4e             else {
05 de                 if (lb) break;
79 05                 x->ch[d] = new node(v);
16 99                 x->ch[d]->p = x;
b4 30                 x = x->ch[d];
60 c2                 break;
88 cb             }
db cb         }
24 0b         splay(x);
c6 61         return lb ? splay(last) : x;
9b cb     }
fd c0     int size() { return root ? root->sz : 0; }
1a 2c     int count(T v) { return insert(v, 1) and root->val == v; }
65 11     node* lower_bound(T v) { return insert(v, 1); }
ae 26     void erase(T v) {
0d 44         if (!count(v)) return;
80 bc         node *x = root, *l = x->ch[0];
8c 26         if (!l) {
bd 8b             root = x->ch[1];
65 32             if (root) root->p = NULL;
e6 8f             return delete x;
9b cb         }
e0 5e         root = l, l->p = NULL;
c0 90         while (l->ch[1]) l = l->ch[1];
bc ba         splay(l);
fd f0         l->ch[1] = x->ch[1];
ad 7d         if (l->ch[1]) l->ch[1]->p = l;
4d bf         delete x;
34 62         l->update();
b7 cb     }
90 24     int order_of_key(T v) {
0d 62         if (!lower_bound(v)) return root ? root->sz : 0;
43 1c         return root->ch[0] ? root->ch[0]->sz : 0;
26 cb     }
c3 db     node* find_by_order(int k) {
7c 08         if (k >= size()) return NULL;
17 52         node* x = root;

```

```

a3 31     while (1) {
7b 20         if (x->ch[0] and x->ch[0]->sz >= k+1) x = x->ch[0];
d0 4e         else {
95 a1             if (x->ch[0]) k -= x->ch[0]->sz;
05 1d             if (!k) return splay(x);
6e eb             k--, x = x->ch[1];
73 cb         }
7f cb     }
e3 cb }
6f 19 T min() {
a5 52     node* x = root;
de 6f     while (x->ch[0]) x = x->ch[0]; // max -> ch[1]
24 3e     return splay(x)->val;
11 cb }
4f 21 };

```

## 6.28 Splay Tree Implicita

```

// vector da NASA
// Um pouco mais rapido q a treap
// O construtor a partir do vector
// eh linear, todas as outras operacoes
// custam O(log(n)) amortizado
// a3575a

```

```

08 08 template<typename T> struct splay {
79 3c     struct node {
0f 18         node *ch[2], *p;
d3 e4         int sz;
9d 87         T val, sub, lazy;
55 aa         bool rev;
1a da         node(T v) {
e4 69             ch[0] = ch[1] = p = NULL;
21 a2             sz = 1;
9f 1e             sub = val = v;
44 c6             lazy = 0;
e7 b6             rev = false;
0e cb         }
88 a9         void prop() {
a8 0e             if (lazy) {
2e 92                 val += lazy, sub += lazy*sz;
49 09                 if (ch[0]) ch[0]->lazy += lazy;
81 1a                 if (ch[1]) ch[1]->lazy += lazy;
04 cb             }
d6 1b             if (rev) {
ec 80                 swap(ch[0], ch[1]);
64 62                 if (ch[0]) ch[0]->rev ^= 1;

```

```

87 ad             if (ch[1]) ch[1]->rev ^= 1;
00 cb         }
e1 a3         lazy = 0, rev = 0;
ea cb     }
43 01     void update() {
9e 0c         sz = 1, sub = val;
2b c7         for (int i = 0; i < 2; i++) if (ch[i]) {
c3 05             ch[i]->prop();
8c d5             sz += ch[i]->sz;
f2 4a             sub += ch[i]->sub;
95 cb         }
6f cb     }
c6 21 };

b0 bb     node* root;

07 5d     splay() { root = NULL; }
eb 9b     splay(node* x) {
80 4e         root = x;
25 32         if (root) root->p = NULL;
51 cb     }
57 1b     splay(vector<T> v) { // O(n)
37 95         root = NULL;
5a 80         for (T i : v) {
4a 2a             node* x = new node(i);
8d bd             x->ch[0] = root;
a8 37             if (root) root->p = x;
2d 4e             root = x;
fb a0             root->update();
66 cb         }
ce cb     }
b2 a9     splay(const splay& t) {
6c e6         throw logic_error("Nao copiar a splay!");
58 cb     }
39 5a     ~splay() {
5c 60         vector<node*> q = {root};
21 40         while (q.size()) {
63 e5             node* x = q.back(); q.pop_back();
5c ee             if (!x) continue;
d9 73             q.push_back(x->ch[0]), q.push_back(x->ch[1]);
21 bf             delete x;
b9 cb         }
f9 cb     }

17 73     int size(node* x) { return x ? x->sz : 0; }
a0 94     void rotate(node* x) { // x vai ficar em cima
85 d9         node *p = x->p, *pp = p->p;

```



```

2a ec     if (pp) pp->ch[pp->ch[1] == p] = x;
14 28     bool d = p->ch[0] == x;
eb d6     p->ch[!d] = x->ch[d], x->ch[d] = p;
c7 ba     if (p->ch[!d]) p->ch[!d]->p = p;
84 fc     x->p = pp, p->p = x;
f9 1e     p->update(), x->update();
f3 cb }
3f 6a node* splaya(node* x) {
88 a3     if (!x) return x;
2e be     root = x, x->update();
6a 3c     while (x->p) {
37 d9         node *p = x->p, *pp = p->p;
d9 35         if (!pp) return rotate(x), x; // zig
67 e3         if ((pp->ch[0] == p)^(p->ch[0] == x))
bf a2             rotate(x), rotate(x); // zigzag
5f 4b         else rotate(p), rotate(x); // zigzig
92 cb     }
9c ea     return x;
92 cb }
d1 a7 node* find(int v) {
c3 a2     if (!root) return NULL;
ce 52     node *x = root;
89 6c     int key = 0;
d7 31     while (1) {
83 85         x->prop();
c8 ba         bool d = key + size(x->ch[0]) < v;
b0 87         if (key + size(x->ch[0]) != v and x->ch[d]) {
42 15             if (d) key += size(x->ch[0])+1;
da 30             x = x->ch[d];
0a 9a         } else break;
aa cb     }
e0 15     return splaya(x);
db cb }
af c0 int size() { return root ? root->sz : 0; }
aa c2 void join(splay<T>& l) { // assume que l < *this
fb 69     if (!size()) swap(root, l.root);
9a 57     if (!size() or !l.size()) return;
3e be     node* x = l.root;
ef 31     while (1) {
c0 85         x->prop();
d1 34         if (!x->ch[1]) break;
9a bd         x = x->ch[1];
45 cb     }
3d 14     l.splaya(x), root->prop(), root->update();
76 42     x->ch[1] = root, x->ch[1]->p = x;
21 0a     root = l.root, l.root = NULL;
63 a0     root->update();

```

```

b5 cb     }
5d 5e node* split(int v) { // retorna os elementos < v
23 39     if (v <= 0) return NULL;
6a 06     if (v >= size()) {
f5 f8         node* ret = root;
5f 95         root = NULL;
af 8c         ret->update();
a0 ed         return ret;
be cb     }
8c ad     find(v);
c0 a5     node* l = root->ch[0];
03 4d     root->ch[0] = NULL;
0a 5a     if (l) l->p = NULL;
b2 a0     root->update();
c1 79     return l;
1b cb }
45 51 T& operator [](int i) {
1c 9d     find(i);
cb ae     return root->val;
d1 cb }
38 23 void push_back(T v) { // O(1)
27 a0     node* r = new node(v);
35 0d     r->ch[0] = root;
83 b1     if (root) root->p = r;
0c b1     root = r, root->update();
8b cb }
4a b7 T query(int l, int r) {
db 95     splay<T> M(split(r+1));
b6 5f     splay<T> L(M.split(l));
39 d1     T ans = M.root->sub;
3e 49     M.join(L), join(M);
86 ba     return ans;
38 cb }
32 41 void update(int l, int r, T s) {
4b 95     splay<T> M(split(r+1));
2b 5f     splay<T> L(M.split(l));
89 99     M.root->lazy += s;
f4 49     M.join(L), join(M);
8c cb }
4b 8c void reverse(int l, int r) {
f7 95     splay<T> M(split(r+1));
11 5f     splay<T> L(M.split(l));
85 94     M.root->rev ^= 1;
e5 49     M.join(L), join(M);
c8 cb }
f3 2f void erase(int l, int r) {
c6 95     splay<T> M(split(r+1));

```

```

91 5f      splay<T> L(M.split(1));
d7 dc      join(L);
01 cb    }
a3 21 };

```

## 6.29 Split-Merge Set

```

// Representa um conjunto de inteiros nao negativos
// Todas as operacoes custam O(log(N)),
// em que N = maior elemento do set,
// exceto o merge, que custa O(log(N)) amortizado
// Usa O(min(N, n log(N))) de memoria, sendo 'n' o
// numero de elementos distintos no set
// 2d2d8a

```

```

2d 2d template<typename T, bool MULTI=false, typename SIZE_T=int>
    struct sms {
a7 3c     struct node {
fb b1         node *l, *r;
9c 15         SIZE_T cnt;
14 65         node() : l(NULL), r(NULL), cnt(0) {}
e0 01         void update() {
4d a0             cnt = 0;
18 d8             if (l) cnt += l->cnt;
fc e4             if (r) cnt += r->cnt;
d0 cb         }
67 21     };

b7 bb     node* root;
5a fd     T N;

41 f3     sms() : root(NULL), N(0) {}
67 83     sms(T v) : sms() { while (v >= N) N = 2*N+1; }
bd 5e     sms(const sms& t) : root(NULL), N(t.N) {
5e 3a         for (SIZE_T i = 0; i < t.size(); i++) {
0d a0             T at = t[i];
0f e6             SIZE_T qt = t.count(at);
1f a4             insert(at, qt);
41 f4             i += qt-1;
73 cb         }
35 cb     }
04 a9     sms(initializer_list<T> v) : sms() { for (T i : v) insert(i); }
10 2d     ~sms() {
aa 60         vector<node*> q = {root};
71 40         while (q.size()) {
0b e5             node* x = q.back(); q.pop_back();
10 ee             if (!x) continue;

```

```

09 1c         q.push_back(x->l), q.push_back(x->r);
c0 bf         delete x;
df cb     }
2c cb }

4e fd     friend void swap(sms& a, sms& b) {
ce 49         swap(a.root, b.root), swap(a.N, b.N);
d9 cb     }
5b 83     sms& operator =(const sms& v) {
b9 76         sms tmp = v;
88 42         swap(tmp, *this);
ec 35         return *this;
79 cb     }
a9 d0     SIZE_T size() const { return root ? root->cnt : 0; }
91 17     SIZE_T count(node* x) const { return x ? x->cnt : 0; }
a5 75     void clear() {
07 0a         sms tmp;
2a 4a         swap(*this, tmp);
fc cb     }
ee a0     void expand(T v) {
50 bc         for (; N < v; N = 2*N+1) if (root) {
c6 63             node* nroot = new node();
41 95             nroot->l = root;
fc 89             root = nroot;
31 a0             root->update();
ac cb         }
f8 cb     }

b1 b1     node* insert(node* at, T idx, SIZE_T qt, T l, T r) {
72 1a         if (!at) at = new node();
22 89         if (l == r) {
e8 43             at->cnt += qt;
10 be             if (!MULTI) at->cnt = 1;
e4 ce             return at;
bd cb         }
23 84         T m = l + (r-l)/2;
b7 a0         if (idx <= m) at->l = insert(at->l, idx, qt, l, m);
57 8d         else at->r = insert(at->r, idx, qt, m+1, r);
53 cf         return at->update(), at;
e2 cb     }
48 cf     void insert(T v, SIZE_T qt=1) { // insere 'qt' ocorrencias de
'v',
82 88         if (qt <= 0) return erase(v, -qt);
4a 72         assert(v >= 0);
16 f5         expand(v);
d5 5e         root = insert(root, v, qt, 0, N);
96 cb     }

```

```

e3 f0 node* erase(node* at, T idx, SIZE_T qt, T l, T r) {
0a 28     if (!at) return at;
20 54     if (l == r) at->cnt = at->cnt < qt ? 0 : at->cnt - qt;
48 4e     else {
fd 84         T m = l + (r-l)/2;
fd 28         if (idx <= m) at->l = erase(at->l, idx, qt, l, m);
da ba         else at->r = erase(at->r, idx, qt, m+1, r);
1a 7b         at->update();
4c cb     }
46 13     if (!at->cnt) delete at, at = NULL;
de ce     return at;
77 cb }
2c 43 void erase(T v, SIZE_T qt=1) { // remove 'qt' ocorrencias de
'v',
ec 9c     if (v < 0 or v > N or !qt) return;
03 9d     if (qt < 0) insert(v, -qt);
1f b1     root = erase(root, v, qt, 0, N);
fd cb }
5c 8d void erase_all(T v) { // remove todos os 'v'
63 34     if (v < 0 or v > N) return;
b2 9f     root = erase(root, v, numeric_limits<SIZE_T>::max(), 0, N);
5f cb }

1f 0f SIZE_T count(node* at, T a, T b, T l, T r) const {
07 61     if (!at or b < l or r < a) return 0;
c9 0f     if (a <= l and r <= b) return at->cnt;
68 84     T m = l + (r-l)/2;
b4 84     return count(at->l, a, b, l, m) + count(at->r, a, b, m+1,
r);
fa cb }
09 0a SIZE_T count(T v) const { return count(root, v, v, 0, N); }
55 ff SIZE_T order_of_key(T v) { return count(root, 0, v-1, 0, N); }
cf df SIZE_T lower_bound(T v) { return order_of_key(v); }

c8 e6 const T operator [] (SIZE_T i) const { // i-esimo menor elemento
f0 80     assert(i >= 0 and i < size());
0a c4     node* at = root;
56 4a     T l = 0, r = N;
1f 40     while (l < r) {
29 84         T m = l + (r-l)/2;
bf 5c         if (count(at->l) > i) at = at->l, r = m;
a0 4e         else {
5a b4             i -= count(at->l);
9c de             at = at->r; l = m+1;
10 cb         }
f5 cb     }

```

```

0f 79     return l;
66 cb }

03 78 node* merge(node* l, node* r) {
89 34     if (!l or !r) return l ? l : r;
76 50     if (!l->l and !l->r) { // folha
0d 59         if (MULTI) l->cnt += r->cnt;
86 55         delete r;
b7 79         return l;
93 cb     }
49 f5     l->l = merge(l->l, r->l), l->r = merge(l->r, r->r);
16 f4     l->update(), delete r;
1c 79     return l;
44 cb }
d0 f5 void merge(sms& s) { // mergeia dois sets
b3 06     if (N > s.N) swap(*this, s);
53 78     expand(s.N);
b2 93     root = merge(root, s.root);
ca ee     s.root = NULL;
8b cb }

c0 dc node* split(node*& x, SIZE_T k) {
93 7c     if (k <= 0 or !x) return NULL;
7c 6d     node* ret = new node();
9f 38     if (!x->l and !x->r) x->cnt -= k, ret->cnt += k;
17 4e     else {
3c 85         if (k <= count(x->l)) ret->l = split(x->l, k);
2c 4e         else {
6f 06             ret->r = split(x->r, k - count(x->l));
76 cf             swap(x->l, ret->l);
76 cb         }
f5 67         ret->update(), x->update();
e1 cb     }
c4 d5     if (!x->cnt) delete x, x = NULL;
00 ed     return ret;
d6 cb }
06 02 void split(SIZE_T k, sms& s) { // pega os 'k' menores
81 e6     s.clear();
1e 6e     s.root = split(root, min(k, size()));
79 e3     s.N = N;
70 cb }
// pega os menores que 'k'
fc 13 void split_val(T k, sms& s) { split(order_of_key(k), s); }
2d 21 };

```

## 6.30 SQRT Tree

```

// RMQ em O(log log n) com O(n log log n) pra buildar
// Funciona com qualquer operacao associativa
// Tao rapido quanto a sparse table, mas usa menos memoria
// (log log (1e9) < 5, entao a query eh praticamente O(1))
//
// build - O(n log log n)
// query - O(log log n)
// 8ff986

97 97 namespace sqrtTree {
3d 05     int n, *v;
0b ec     int pref[4][MAX], sulf[4][MAX], getl[4][MAX], entre[4][MAX],
          sz[4];

df 5f     int op(int a, int b) { return min(a, b); }
66 c7     inline int getblk(int p, int i) { return (i-getl[p][i])/sz[p];
          }
89 2c     void build(int p, int l, int r) {
97 bc         if (l+1 >= r) return;
0d 36         for (int i = l; i <= r; i++) getl[p][i] = l;
da f1         for (int L = l; L <= r; L += sz[p]) {
14 19             int R = min(L+sz[p]-1, r);
68 89             pref[p][L] = v[L], sulf[p][R] = v[R];
2f 59             for (int i = L+1; i <= R; i++) pref[p][i] =
op(pref[p][i-1], v[i]);
e4 d9             for (int i = R-1; i >= L; i--) sulf[p][i] = op(v[i],
sulf[p][i+1]);
ca 22             build(p+1, L, R);
9b cb         }
4f 69         for (int i = 0; i <= sz[p]; i++) {
c7 ca             int at = entre[p][l+i*sz[p]+i] = sulf[p][l+i*sz[p]];
c2 75             for (int j = i+1; j <= sz[p]; j++)
entre[p][l+i*sz[p]+j] = at =
47 23                 op(at, sulf[p][l+j*sz[p]]);
20 cb         }
b4 cb     }
0f 0d     void build(int n2, int* v2) {
ab 68         n = n2, v = v2;
6c 44         for (int p = 0; p < 4; p++) sz[p] = n2 = sqrt(n2);
bd c5         build(0, 0, n-1);
15 cb     }
65 9e     int query(int l, int r) {
81 79         if (l+1 >= r) return l == r ? v[l] : op(v[l], v[r]);
57 1b         int p = 0;
42 4b         while (getblk(p, l) == getblk(p, r)) p++;
9e 9e         int ans = sulf[p][l], a = getblk(p, l)+1, b = getblk(p,
r)-1;

```

```

ba 8b         if (a <= b) ans = op(ans, entre[p][getl[p][l]+a*sz[p]+b]);
7e de         return op(ans, pref[p][r]);
8a cb     }
8f cb }

```

## 6.31 Treap

```

// Todas as operacoes custam
// O(log(n)) com alta probabilidade, exceto meld
// meld custa O(log^2 n) amortizado com alta prob.,
// e permite unir duas treaps sem restricao adicional
// Na pratica, esse meld tem constante muito boa e
// o pior caso eh meio estranho de acontecer
// bd93e2

87 87 mt19937 rng((int)
chrono::steady_clock::now().time_since_epoch().count());

9e aa template<typename T> struct treap {
35 3c     struct node {
ee b1         node *l, *r;
54 28         int p, sz;
f2 36         T val, mi;
e1 4c         node(T v) : l(NULL), r(NULL), p(rng()), sz(1), val(v),
mi(v) {}
b4 01         void update() {
34 a2             sz = 1;
7c d6             mi = val;
cc bd             if (l) sz += l->sz, mi = min(mi, l->mi);
cf a5             if (r) sz += r->sz, mi = min(mi, r->mi);
ee cb         }
94 21     };

94 bb     node* root;

52 84     treap() { root = NULL; }
0d 2d     treap(const treap& t) {
2c 46         throw logic_error("Nao copiar a treap!");
03 cb     }
5c ce     ~treap() {
30 60         vector<node*> q = {root};
f5 40         while (q.size()) {
89 e5             node* x = q.back(); q.pop_back();
7f ee             if (!x) continue;
2b 1c             q.push_back(x->l), q.push_back(x->r);
8a bf             delete x;
7a cb         }

```

```

b9 cb }

8d 73 int size(node* x) { return x ? x->sz : 0; }
29 b2 int size() { return size(root); }
9f bc void join(node* l, node* r, node*& i) { // assume que l < r
5a 98     if (!l or !r) return void(i = l ? l : r);
73 80     if (l->p > r->p) join(l->r, r, l->r), i = l;
4e fa     else join(l, r->l, r->l), i = r;
d6 bd     i->update();
5b cb }
8e ec void split(node* i, node*& l, node*& r, T v) {
2d 26     if (!i) return void(r = l = NULL);
16 f0     if (i->val < v) split(i->r, i->r, r, v), l = i;
d8 80     else split(i->l, l, i->l, v), r = i;
ad bd     i->update();
8c cb }
c2 3f void split_leq(node* i, node*& l, node*& r, T v) {
2d 26     if (!i) return void(r = l = NULL);
4c 18     if (i->val <= v) split_leq(i->r, i->r, r, v), l = i;
70 58     else split_leq(i->l, l, i->l, v), r = i;
48 bd     i->update();
c7 cb }
4c e1 int count(node* i, T v) {
47 6b     if (!i) return 0;
e7 35     if (i->val == v) return 1;
86 8d     if (v < i->val) return count(i->l, v);
c1 4d     return count(i->r, v);
cb cb }
5b 26 void index_split(node* i, node*& l, node*& r, int v, int key =
0) {
39 26     if (!i) return void(r = l = NULL);
5b c1     if (key + size(i->l) < v) index_split(i->r, i->r, r, v,
key+size(i->l)+1), l = i;
74 e5     else index_split(i->l, l, i->l, v, key), r = i;
73 bd     i->update();
ec cb }
4e a1 int count(T v) {
e3 e0     return count(root, v);
a6 cb }
a6 c2 void insert(T v) {
e3 98     if (count(v)) return;
1b 03     node *L, *R;
82 d4     split(root, L, R, v);
2a 58     node* at = new node(v);
89 59     join(L, at, L);
46 a2     join(L, R, root);
4d cb }

```

```

7d 26 void erase(T v) {
2f df     node *L, *M, *R;
bf b6     split_leq(root, M, R, v), split(M, L, M, v);
0c f1     if (M) delete M;
c1 f3     M = NULL;
68 a2     join(L, R, root);
64 cb }
7a e7 void meld(treap& t) { // segmented merge
60 4a     node *L = root, *R = t.root;
26 95     root = NULL;
e8 6b     while (L or R) {
13 fe         if (!L or (L and R and L->mi > R->mi)) std::swap(L, R);
76 5e         if (!R) join(root, L, root), L = NULL;
0e 3c         else if (L->mi == R->mi) {
a7 a7             node* LL;
3f 43             split(L, LL, L, R->mi+1);
b0 35             delete LL;
17 9d         } else {
aa a7             node* LL;
2b 53             split(L, LL, L, R->mi);
da db             join(root, LL, root);
bd cb         }
13 cb     }
06 68     t.root = NULL;
aa cb }
bd 21 };

```

## 6.32 Treap Implicita

```

// Todas as operacoes custam
// O(log(n)) com alta probabilidade
// 63ba4d

87 87 mt19937 rng((int)
chrono::steady_clock::now().time_since_epoch().count());

9e aa template<typename T> struct treap {
35 3c     struct node {
ee b1         node *l, *r;
54 28         int p, sz;
82 87         T val, sub, lazy;
d4 aa         bool rev;
b2 8d         node(T v) : l(NULL), r(NULL), p(rng()), sz(1), val(v),
sub(v), lazy(0), rev(0) {}
ac a9         void prop() {
f7 0e             if (lazy) {
48 92                 val += lazy, sub += lazy*sz;

```

```

f4 b8         if (l) l->lazy += lazy;
e9 d3         if (r) r->lazy += lazy;
77 cb     }
b5 1b     if (rev) {
f7 e4         swap(l, r);
a4 dc         if (l) l->rev ^= 1;
8d f2         if (r) r->rev ^= 1;
12 cb     }
d7 a3         lazy = 0, rev = 0;
e8 cb     }
bc 01     void update() {
2e 0c         sz = 1, sub = val;
20 a0         if (l) l->prop(), sz += l->sz, sub += l->sub;
27 09         if (r) r->prop(), sz += r->sz, sub += r->sub;
9f cb     }
a6 21 };

fa bb     node* root;

e6 84     treap() { root = NULL; }
77 2d     treap(const treap& t) {
09 46         throw logic_error("Nao copiar a treap!");
21 cb     }
8e ce     ~treap() {
27 60         vector<node*> q = {root};
0e 40         while (q.size()) {
74 e5             node* x = q.back(); q.pop_back();
84 ee             if (!x) continue;
bc 1c             q.push_back(x->l), q.push_back(x->r);
a3 bf             delete x;
c6 cb         }
c2 cb     }

2c 73     int size(node* x) { return x ? x->sz : 0; }
48 b2     int size() { return size(root); }
7a bc     void join(node* l, node* r, node*& i) { // assume que l < r
07 98         if (!l or !r) return void(i = l ? l : r);
75 16         l->prop(), r->prop();
e8 80         if (l->p > r->p) join(l->r, r, l->r), i = l;
07 fa         else join(l, r->l, r->l), i = r;
4b bd         i->update();
d7 cb     }
d0 a2     void split(node* i, node*& l, node*& r, int v, int key = 0) {
53 26         if (!i) return void(r = l = NULL);
c2 c8         i->prop();
ff 5b         if (key + size(i->l) < v) split(i->r, i->r, r, v,
key+size(i->l)+1), l = i;

```

```

b7 21         else split(i->l, l, i->l, v, key), r = i;
af bd         i->update();
1f cb     }
ab 23     void push_back(T v) {
a3 2e         node* i = new node(v);
0f 7a         join(root, i, root);
9b cb     }
9b b7     T query(int l, int r) {
42 df         node *L, *M, *R;
1b dc         split(root, M, R, r+1), split(M, L, M, l);
d8 d4         T ans = M->sub;
8d 69         join(L, M, M), join(M, R, root);
79 ba         return ans;
79 cb     }
a1 41     void update(int l, int r, T s) {
96 df         node *L, *M, *R;
62 dc         split(root, M, R, r+1), split(M, L, M, l);
1a 8f         M->lazy += s;
26 69         join(L, M, M), join(M, R, root);
cd cb     }
7f 8c     void reverse(int l, int r) {
63 df         node *L, *M, *R;
ae dc         split(root, M, R, r+1), split(M, L, M, l);
1d 66         M->rev ^= 1;
0e 69         join(L, M, M), join(M, R, root);
55 cb     }
63 21 };

```

### 6.33 Treap Persistent Implicita

```

// Todas as operacoes custam
// O(log(n)) com alta probabilidade
// fb8013

6c 6c mt19937_64 rng((int)
chrono::steady_clock::now().time_since_epoch().count());

ef 3c struct node {
f7 b1     node *l, *r;
5a f1     ll sz, val, sub;
21 30     node(ll v) : l(NULL), r(NULL), sz(1), val(v), sub(v) {}
9b c1     node(node* x) : l(x->l), r(x->r), sz(x->sz), val(x->val),
sub(x->sub) {}
06 01     void update() {
d4 0c         sz = 1, sub = val;
da 77         if (l) sz += l->sz, sub += l->sub;
0c d6         if (r) sz += r->sz, sub += r->sub;

```

```

af 12      sub %= MOD;
c6 cb    }
f7 21 };

b7 bc 11 size(node* x) { return x ? x->sz : 0; }
69 76 void update(node* x) { if (x) x->update(); }
61 82 node* copy(node* x) { return x ? new node(x) : NULL; }

ea b0 node* join(node* l, node* r) {
97 e1   if (!l or !r) return l ? copy(l) : copy(r);
0a 48   node* ret;
c5 49   if (rng() % (size(l) + size(r)) < size(l)) {
ec 7e       ret = copy(l);
c1 cc       ret->r = join(ret->r, r);
27 9d   } else {
ac 4c       ret = copy(r);
5a 55       ret->l = join(l, ret->l);
e6 cb   }
be 74   return update(ret), ret;
f2 cb }

c1 72 void split(node* x, node*& l, node*& r, ll v, ll key = 0) {
de 42   if (!x) return void(l = r = NULL);
e4 b4   if (key + size(x->l) < v) {
f8 72       l = copy(x);
06 d7       split(l->r, l->r, r, v, key+size(l->l)+1);
cb 9d   } else {
a9 30       r = copy(x);
ab 41       split(r->l, l, r->l, v, key);
59 cb   }
e3 da   update(l), update(r);
7d cb }

c9 f9 vector<node*> treap;

6c 13 void init(const vector<ll>& v) {
11 bb   treap = {NULL};
ff 96   for (auto i : v) treap[0] = join(treap[0], new node(i));
fb cb }

```

## 6.34 Wavelet Tree

```

// Usa O(sigma + n log(sigma)) de memoria,
// onde sigma = MAXN - MINN
// Depois do build, o v fica ordenado
// count(i, j, x, y) retorna o numero de elementos de
// v[i, j) que pertencem a [x, y]

```

```

// kth(i, j, k) retorna o elemento que estaria
// na posicao k-1 de v[i, j), se ele fosse ordenado
// sum(i, j, x, y) retorna a soma dos elementos de
// v[i, j) que pertencem a [x, y]
// sumk(i, j, k) retorna a soma dos k-esimos menores
// elementos de v[i, j) (sum(i, j, 1) retorna o menor)
//
// Complexidades:
// build - O(n log(sigma))
// count - O(log(sigma))
// kth - O(log(sigma))
// sum - O(log(sigma))
// sumk - O(log(sigma))
// 782344

59 59 int n, v[MAXN];
b5 57 vector<int> esq[4*(MAXN-MINN)], pref[4*(MAXN-MINN)];

d7 f8 void build(int b = 0, int e = n, int p = 1, int l = MINN, int r = MAXN) {
9c 58     int m = (l+r)/2; esq[p].push_back(0); pref[p].push_back(0);
0e f2     for (int i = b; i < e; i++) {
0b 6b         esq[p].push_back(esq[p].back()+(v[i]<=m));
da 26         pref[p].push_back(pref[p].back()+v[i]);
3d cb     }
ba 8c     if (l == r) return;
89 3a     int m2 = stable_partition(v+b, v+e, [=](int i){return i <= m;}) - v;
35 34     build(b, m2, 2*p, l, m), build(m2, e, 2*p+1, m+1, r);
b0 cb }

e0 54 int count(int i, int j, int x, int y, int p = 1, int l = MINN, int r = MAXN) {
39 2a     if (y < l or r < x) return 0;
ab 4d     if (x <= l and r <= y) return j-i;
b9 dd     int m = (l+r)/2, ei = esq[p][i], ej = esq[p][j];
48 0a     return count(ei, ej, x, y, 2*p, l, m)+count(i-ei, j-ej, x, y, 2*p+1, m+1, r);
f8 cb }

1f f6 int kth(int i, int j, int k, int p=1, int l = MINN, int r = MAXN) {
a4 3c     if (l == r) return l;
b3 dd     int m = (l+r)/2, ei = esq[p][i], ej = esq[p][j];
13 58     if (k <= ej-ei) return kth(ei, ej, k, 2*p, l, m);
92 28     return kth(i-ei, j-ej, k-(ej-ei), 2*p+1, m+1, r);
08 cb }

```

```

68 f2 int sum(int i, int j, int x, int y, int p = 1, int l = MINN, int
    r = MAXN) {
3a 2a     if (y < l or r < x) return 0;
68 2a     if (x <= l and r <= y) return pref[p][j]-pref[p][i];
38 dd     int m = (l+r)/2, ei = esq[p][i], ej = esq[p][j];
0f 43     return sum(ei, ej, x, y, 2*p, l, m) + sum(i-ei, j-ej, x, y,
        2*p+1, m+1, r);
ad cb }

ac b8 int sumk(int i, int j, int k, int p = 1, int l = MINN, int r =
    MAXN) {
d4 8a     if (l == r) return l*k;
79 dd     int m = (l+r)/2, ei = esq[p][i], ej = esq[p][j];
37 50     if (k <= ej-ei) return sumk(ei, ej, k, 2*p, l, m);
a6 4c     return pref[2*p][ej]-pref[2*p][ei]+sumk(i-ei, j-ej, k-(ej-ei),
        2*p+1, m+1, r);
78 cb }

```

## 7 Grafos

### 7.1 AGM Direcionada

```

// Fala o menor custo para selecionar arestas tal que
// o vertice 'r' alcance todos
// Se nao tem como, retorna LINF
//
// O(m log(n))
// dc345b

3c 3c struct node {
36 f3     pair<ll, int> val;
2a 4e     ll lazy;
2e b1     node *l, *r;
93 f9     node() {}
e5 c5     node(pair<int, int> v) : val(v), lazy(0), l(NULL), r(NULL) {}

2f a9     void prop() {
18 76         val.first += lazy;
51 b8         if (l) l->lazy += lazy;
aa d3         if (r) r->lazy += lazy;
73 c6         lazy = 0;
4a cb     }
29 21 };
59 de void merge(node*& a, node* b) {
f9 c1     if (!a) swap(a, b);

```

```

00 80     if (!b) return;
24 62     a->prop(), b->prop();
68 d0     if (a->val > b->val) swap(a, b);
fb 4b     merge(rand()%2 ? a->l : a->r, b);
8f cb }

dc d0 pair<ll, int> pop(node*& R) {
ea e8     R->prop();
bb 22     auto ret = R->val;
90 af     node* tmp = R;
6b 3f     merge(R->l, R->r);
8e 6c     R = R->l;
23 3e     if (R) R->lazy -= ret.first;
d1 7c     delete tmp;
5b ed     return ret;
a1 cb }

17 6f void apaga(node* R) { if (R) apaga(R->l), apaga(R->r), delete R;
    }

da f1 ll dmst(int n, int r, vector<pair<pair<int, int>, int>>& ar) {
90 94     vector<int> p(n); iota(p.begin(), p.end(), 0);
0b a2     function<int(int)> find = [&](int k) { return
        p[k]==k?k:p[k]=find(p[k]); };
72 2d     vector<node*> h(n);
cf 56     for (auto e : ar) merge(h[e.first.second], new node({e.second,
        e.first.first}));
87 fd     vector<int> pai(n, -1), path(n);
33 66     pai[r] = r;
b8 04     ll ans = 0;

13 60     for (int i = 0; i < n; i++) { // vai conectando todo mundo
03 2a         int u = i, at = 0;
44 ca         while (pai[u] == -1) {
b4 da             if (!h[u]) { // nao tem
b0 94                 for (auto i : h) apaga(i);
6c 77                 return LINF;
e3 cb             }
0b 16             path[at++] = u, pai[u] = i;
f5 55             auto [mi, v] = pop(h[u]);
16 64             ans += mi;

06 5e             if (pai[u = find(v)] == i) { // ciclo
04 86                 while (find(v = path[--at]) != u)
b7 62                     merge(h[u], h[v]), h[v] = NULL, p[find(v)] = u;
03 57                 pai[u] = -1;
2c cb             }
63 cb         }
6e cb     }

```



```

d4 94     for (auto i : h) apaga(i);
5b ba     return ans;
dc cb }

```

## 7.2 Articulation Points

```

// Computa os pontos de articulacao (vertices criticos) de um grafo
//
// art[i] armazena o numero de novas componentes criadas ao deletar
//     vertice i
// se art[i] >= 1, entao vertice i eh ponto de articulacao
//
// 0(n+m)
// 0e405b

```

```

1a 1a int n;
c4 78 vector<vector<int>>> g;
f8 4c stack<int> s;
ea b6 vector<int> id, art;

```

```

4c 3e int dfs_art(int i, int& t, int p = -1) {
dc cf     int lo = id[i] = t++;
3b 18     s.push(i);
15 ca     for (int j : g[i]) if (j != p) {
8c 9a         if (id[j] == -1) {
d8 20             int val = dfs_art(j, t, i);
85 0c             lo = min(lo, val);

a4 58             if (val >= id[i]) {
d7 66                 art[i]++;
fd bd                 while (s.top() != j) s.pop();
0b 2e                 s.pop();
34 cb             }

// if (val > id[i]) aresta i-j eh ponte

d6 cb     }
7a 32     else lo = min(lo, id[j]);
bb cb     }
d0 3b     if (p == -1 and art[i]) art[i]--;
70 25     return lo;
59 cb }

```

```

69 d7 void compute_art_points() {
18 59     id = vector<int>(n, -1);
dd a6     art = vector<int>(n, 0);
25 6b     int t = 0;
0e d4     for (int i = 0; i < n; i++) if (id[i] == -1)
eb 62         dfs_art(i, t, -1);

```

```

0e cb }

```

## 7.3 Bellman-Ford

```

// Calcula a menor distancia
// entre a e todos os vertices e
// detecta ciclo negativo
// Retorna 1 se ha ciclo negativo
// Nao precisa representar o grafo,
// soh armazenar as arestas
//
// 0(nm)
// 03059b

14 14 int n, m;
17 24 int d[MAX];
de e9 vector<pair<int, int>> ar; // vetor de arestas
df 9e vector<int> w;           // peso das arestas

```

```

6b 6b bool bellman_ford(int a) {
34 8e     for (int i = 0; i < n; i++) d[i] = INF;
a9 8a     d[a] = 0;

15 4e     for (int i = 0; i <= n; i++)
f7 89         for (int j = 0; j < m; j++) {
61 6e             if (d[ar[j].second] > d[ar[j].first] + w[j]) {
d8 70                 if (i == n) return 1;

b8 e9                 d[ar[j].second] = d[ar[j].first] + w[j];
a5 cb             }
57 cb         }

e7 bb     return 0;
03 cb }

```

## 7.4 Block-Cut Tree

```

// Cria a block-cut tree, uma arvore com os blocos
// e os pontos de articulacao
// Blocos sao componentes 2-vertice-conexos maximais
// Uma 2-coloracao da arvore eh tal que uma cor sao
// os blocos, e a outra cor sao os pontos de art.
// Funciona para grafo nao conexo
//
// art[i] responde o numero de novas componentes conexas
// criadas apos a remocao de i do grafo g
// Se art[i] >= 1, i eh ponto de articulacao

```

```

//
// Para todo i <= blocks.size()
// blocks[i] eh uma componente 2-vertice-conexa maximal
// edgblocks[i] sao as arestas do bloco i
// tree[i] eh um vertice da arvore que corresponde ao bloco i
//
// pos[i] responde a qual vertice da arvore vertice i pertence
// Arvore tem no maximo 2n vertices
//
// O(n+m)
// 056fa2

d1 d1 struct block_cut_tree {
e1 d8     vector<vector<int>> g, blocks, tree;
8b 43     vector<vector<pair<int, int>>> edgblocks;
a2 4c     stack<int> s;
4a 6c     stack<pair<int, int>> s2;
66 2b     vector<int> id, art, pos;

78 76     block_cut_tree(vector<vector<int>> g_) : g(g_) {
4d af         int n = g.size();
c3 37         id.resize(n, -1), art.resize(n), pos.resize(n);
29 6f         build();
da cb     }

7f df     int dfs(int i, int& t, int p = -1) {
5a cf         int lo = id[i] = t++;
07 18         s.push(i);

1f 82         if (p != -1) s2.emplace(i, p);
3c 53         for (int j : g[i]) if (j != p and id[j] != -1)
            s2.emplace(i, j);

dc ca         for (int j : g[i]) if (j != p) {
3f 9a             if (id[j] == -1) {
43 12                 int val = dfs(j, t, i);
c8 0c                 lo = min(lo, val);

9b 58                 if (val >= id[i]) {
3d 66                     art[i]++;
e4 48                     blocks.emplace_back(1, i);
2d 11                     while (blocks.back().back() != j)
c5 13                         blocks.back().push_back(s.top()), s.pop();

00 12                     edgblocks.emplace_back(1, s2.top()), s2.pop();
40 47                     while (edgblocks.back().back() != pair(j, i))
65 bc                         edgblocks.back().push_back(s2.top()),

```

```

            s2.pop();
cd cb                }
                        // if (val > id[i]) aresta i-j eh ponte
30 cb                }
8f 32                else lo = min(lo, id[j]);
fb cb                }

6c 3b                if (p == -1 and art[i]) art[i]--;
fe 25                return lo;
8f cb                }

e9 0a     void build() {
97 6b         int t = 0;
c7 ab         for (int i = 0; i < g.size(); i++) if (id[i] == -1) dfs(i,
            t, -1);

d5 56         tree.resize(blocks.size());
ce f7         for (int i = 0; i < g.size(); i++) if (art[i])
a1 96             pos[i] = tree.size(), tree.emplace_back();

2a 97         for (int i = 0; i < blocks.size(); i++) for (int j :
            blocks[i]) {
16 40             if (!art[j]) pos[j] = i;
c8 10             else tree[i].push_back(pos[j]),
                tree[pos[j]].push_back(i);
92 cb                }
6f cb                }
05 21 };

```

## 7.5 Blossom - matching maximo em grafo geral

```

// O(n^3)
// Se for bipartido, nao precisa da funcao
// 'contract', e roda em O(nm)
// 4426a4

04 04 vector<int> g[MAX];
c2 12 int match[MAX]; // match[i] = com quem i esta matchzado ou -1
cf 1f int n, pai[MAX], base[MAX], vis[MAX];
ec 26 queue<int> q;

a9 10 void contract(int u, int v, bool first = 1) {
64 16     static vector<bool> blossom;
5a fb     static int l;
38 41     if (first) {
11 a4         blossom = vector<bool>(n, 0);
9e 04         vector<bool> teve(n, 0);

```

```

61 dd     int k = u; l = v;
bf 31     while (1) {
d1 29         teve[k = base[k]] = 1;
d3 11         if (match[k] == -1) break;
dd df         k = pai[match[k]];
a2 cb     }
4c d3     while (!teve[l = base[l]]) l = pai[match[l]];
80 cb }
73 2e     while (base[u] != 1) {
45 e2         blossom[base[u]] = blossom[base[match[u]]] = 1;
a1 8f         pai[u] = v;
8a 0b         v = match[u];
3a a5         u = pai[match[u]];
28 cb     }
4e 71     if (!first) return;
a2 95     contract(v, u, 0);
86 6e     for (int i = 0; i < n; i++) if (bloss[base[i]]) {
bc 59         base[i] = 1;
e9 ca         if (!vis[i]) q.push(i);
b9 29         vis[i] = 1;
ab cb     }
b8 cb }

5a f1 int getpath(int s) {
02 88     for (int i = 0; i < n; i++) base[i] = i, pai[i] = -1, vis[i] =
0;
66 de     vis[s] = 1; q = queue<int>(); q.push(s);
bb 40     while (q.size()) {
25 be         int u = q.front(); q.pop();
cd bd         for (int i : g[u]) {
11 7a             if (base[i] == base[u] or match[u] == i) continue;
f6 e3             if (i == s or (match[i] != -1 and pai[match[i]] != -1))
b0 4f                 contract(u, i);
b6 e2             else if (pai[i] == -1) {
d4 54                 pai[i] = u;
50 f6                 if (match[i] == -1) return i;
bf 81                 i = match[i];
dc 29                 vis[i] = 1; q.push(i);
12 cb             }
a5 cb         }
0e cb     }
94 da     return -1;
77 cb }

9f 83 int blossom() {
53 1a     int ans = 0;
2e 31     memset(match, -1, sizeof(match));

```

```

d2 2e     for (int i = 0; i < n; i++) if (match[i] == -1)
0a f7         for (int j : g[i]) if (match[j] == -1) {
9c 1b             match[i] = j;
77 f1             match[j] = i;
27 0d             ans++;
8c c2             break;
a6 cb         }
f3 da     for (int i = 0; i < n; i++) if (match[i] == -1) {
fe 7e         int j = getpath(i);
16 5f         if (j == -1) continue;
20 0d         ans++;
ca 3a         while (j != -1) {
d2 ef             int p = pai[j], pp = match[p];
29 34             match[p] = j;
f0 fe             match[j] = p;
1b 55             j = pp;
38 cb         }
6c cb     }
89 ba     return ans;
44 cb }

```

## 7.6 Centro de arvore

```

// Retorna o diametro e o(s) centro(s) da arvore
// Uma arvore tem sempre um ou dois centros e estes estao no meio do
// diametro
//
// O(n)
// cladeb

04 04 vector<int> g[MAX];
46 df int d[MAX], par[MAX];

30 54 pair<int, vector<int>> center() {
3f a9     int f, df;
d5 36     function<void(int)> dfs = [&] (int v) {
25 d4         if (d[v] > df) f = v, df = d[v];
c1 e6         for (int u : g[v]) if (u != par[v])
a1 1a             d[u] = d[v] + 1, par[u] = v, dfs(u);
25 21     };

ba 1b     f = df = par[0] = -1, d[0] = 0;
17 41     dfs(0);
cc c2     int root = f;
7f 0f     f = df = par[root] = -1, d[root] = 0;
2f 14     dfs(root);

```

```

99 76     vector<int> c;
c6 87     while (f != -1) {
d9 99         if (d[f] == df/2 or d[f] == (df+1)/2) c.push_back(f);
8f 19         f = par[f];
68 cb     }

0e 00     return {df, c};
c1 cb }

```

## 7.7 Centroid

```

// Computa os 2 centroids da arvore
//
// O(n)
// e16075

97 97 int n, subsize[MAX];
2a 04 vector<int> g[MAX];

13 98 void dfs(int k, int p=-1) {
f2 bd     subsize[k] = 1;
25 6e     for (int i : g[k]) if (i != p) {
f5 80         dfs(i, k);
c9 2e         subsize[k] += subsize[i];
06 cb     }
4f cb }

cf 2e int centroid(int k, int p=-1, int size=-1) {
db e7     if (size == -1) size = subsize[k];
8c 8d     for (int i : g[k]) if (i != p) if (subsize[i] > size/2)
9a ba         return centroid(i, k, size);
a8 83     return k;
65 cb }

bd f2 pair<int, int> centroids(int k=0) {
84 05     dfs(k);
95 90     int i = centroid(k), i2 = i;
f1 8d     for (int j : g[i]) if (2*subsize[j] == subsize[k]) i2 = j;
be 0c     return {i, i2};
e1 cb }

```

## 7.8 Centroid decomposition

```

// decomp(0, k) computa numero de caminhos com 'k' arestas
// Mudar depois do comentario
//
// O(n log(n))

```

```

// fe2541

04 04 vector<int> g[MAX];
d8 ba int sz[MAX], rem[MAX];

6a 74 void dfs(vector<int>& path, int i, int l=-1, int d=0) {
93 54     path.push_back(d);
a1 75     for (int j : g[i]) if (j != l and !rem[j]) dfs(path, j, i,
d+1);
28 cb }

02 07 int dfs_sz(int i, int l=-1) {
30 02     sz[i] = 1;
48 e5     for (int j : g[i]) if (j != l and !rem[j]) sz[i] += dfs_sz(j,
i);
4f 19     return sz[i];
71 cb }

1b 85 int centroid(int i, int l, int size) {
4e 99     for (int j : g[i]) if (j != l and !rem[j] and sz[j] > size / 2)
04 73         return centroid(j, i, size);
5a d9     return i;
d8 cb }

93 d7 11 decomp(int i, int k) {
52 10     int c = centroid(i, i, dfs_sz(i));
7d a6     rem[c] = 1;

// gasta O(n) aqui - dfs sem ir pros caras removidos
49 04     11 ans = 0;
05 02     vector<int> cnt(sz[i]);
0f 87     cnt[0] = 1;
cf 0a     for (int j : g[c]) if (!rem[j]) {
7e 5b         vector<int> path;
d7 ba         dfs(path, j);
07 1a         for (int d : path) if (0 <= k-d-1 and k-d-1 < sz[i])
fc 28             ans += cnt[k-d-1];
db e8         for (int d : path) cnt[d+1]++;
21 cb     }

30 1c     for (int j : g[c]) if (!rem[j]) ans += decomp(j, k);
2d 3f     rem[c] = 0;
31 ba     return ans;
fe cb }

```

## 7.9 Centroid Tree

```

// Constroi a centroid tree
// p[i] eh o pai de i na centroid-tree
// dist[i][k] = distancia na arvore original entre i
// e o k-esimo ancestral na arvore da centroid
//
// O(n log(n)) de tempo e memoria
// a0e7c7

84 84 vector<int> g[MAX], dist[MAX];
59 c1 int sz[MAX], rem[MAX], p[MAX];

94 07 int dfs_sz(int i, int l=-1) {
a0 02     sz[i] = 1;
c5 e5     for (int j : g[i]) if (j != l and !rem[j]) sz[i] += dfs_sz(j,
i);
28 19     return sz[i];
11 cb }

9d 85 int centroid(int i, int l, int size) {
2f 99     for (int j : g[i]) if (j != l and !rem[j] and sz[j] > size / 2)
24 73         return centroid(j, i, size);
a9 d9     return i;
26 cb }

33 32 void dfs_dist(int i, int l, int d=0) {
7d 54     dist[i].push_back(d);
81 5a     for (int j : g[i]) if (j != l and !rem[j])
0e 82         dfs_dist(j, i, d+1);
86 cb }

87 27 void decomp(int i, int l = -1) {
53 10     int c = centroid(i, i, dfs_sz(i));
a0 1b     rem[c] = 1, p[c] = l;
02 53     dfs_dist(c, c);
aa a2     for (int j : g[c]) if (!rem[j]) decomp(j, c);
bc cb }

c2 76 void build(int n) {
82 23     for (int i = 0; i < n; i++) rem[i] = 0, dist[i].clear();
c0 86     decomp(0);
31 96     for (int i = 0; i < n; i++) reverse(dist[i].begin(),
dist[i].end());
a0 cb }

```

## 7.10 Dijkstra

```

// encontra menor distancia de x

```

```

// para todos os vertices
// se ao final do algoritmo d[i] = LINF,
// entao x nao alcanca i
//
// O(m log(n))
// 695ac4

ef ef ll d[MAX];
81 c0 vector<pair<int, int>> g[MAX]; // {vizinho, peso}

a1 1a int n;

fc ab void dijkstra(int v) {
46 22     for (int i = 0; i < n; i++) d[i] = LINF;
c1 a7     d[v] = 0;
72 88     priority_queue<pair<ll, int>> pq;
8b b3     pq.emplace(0, v);

26 26     while (pq.size()) {
47 a2         auto [ndist, u] = pq.top(); pq.pop();
52 95         if (-ndist > d[u]) continue;

28 cd         for (auto [idx, w] : g[u]) if (d[idx] > d[u] + w) {
d2 33             d[idx] = d[u] + w;
5e a8             pq.emplace(-d[idx], idx);
e4 cb         }
22 cb     }
69 cb }

7.11 Dinitz

// O(min(m * max_flow, n^2 m))
// Grafo com capacidades 1: O(min(m sqrt(m), m * n^(2/3)))
// Todo vertice tem grau de entrada ou saida 1: O(m sqrt(n))

// 67ce89
47 47 struct dinitz {
d7 61     const bool scaling = false; // com scaling -> O(nm
log(MAXCAP)),
2b 20     int lim; // com constante alta
ea 67     struct edge {
73 35         int to, cap, rev, flow;
f8 7f         bool res;
78 d3         edge(int to_, int cap_, int rev_, bool res_)
64 a9             : to(to_), cap(cap_), rev(rev_), flow(0), res(res_) {}
68 21     };

```

```

02 00    vector<vector<edge>> g;
6b 21    vector<int> lev, beg;
54 a7    ll F;
ca 19    dinitz(int n) : g(n), F(0) {}

04 08    void add(int a, int b, int c) {
a6 ba        g[a].emplace_back(b, c, g[b].size(), false);
28 4c        g[b].emplace_back(a, 0, g[a].size()-1, true);
53 cb    }
9e 12    bool bfs(int s, int t) {
22 90        lev = vector<int>(g.size(), -1); lev[s] = 0;
ec 64        beg = vector<int>(g.size(), 0);
09 8b        queue<int> q; q.push(s);
cb 40        while (q.size()) {
3d be            int u = q.front(); q.pop();
90 bd            for (auto& i : g[u]) {
be db                if (lev[i.to] != -1 or (i.flow == i.cap)) continue;
c4 b4                if (scaling and i.cap - i.flow < lim) continue;
5e 18                lev[i.to] = lev[u] + 1;
72 8c                q.push(i.to);
18 cb            }
e3 cb        }
34 0d        return lev[t] != -1;
40 cb    }
d5 df    int dfs(int v, int s, int f = INF) {
e2 50        if (!f or v == s) return f;
10 88        for (int& i = beg[v]; i < g[v].size(); i++) {
83 02            auto& e = g[v][i];
56 20            if (lev[e.to] != lev[v] + 1) continue;
10 ee            int foi = dfs(e.to, s, min(f, e.cap - e.flow));
9d 74            if (!foi) continue;
7c 3c            e.flow += foi, g[e.to][e.rev].flow -= foi;
49 45            return foi;
04 cb        }
75 bb        return 0;
87 cb    }
a2 ff    ll max_flow(int s, int t) {
f6 a8        for (lim = scaling ? (1<<30) : 1; lim; lim /= 2)
46 9d            while (bfs(s, t)) while (int ff = dfs(s, t)) F += ff;
76 4f        return F;
c8 cb    }
86 21 };

// Recupera as arestas do corte s-t
// d23977
1f db    vector<pair<int, int>> get_cut(dinitz& g, int s, int t) {
93 f0        g.max_flow(s, t);

```

```

20 68    vector<pair<int, int>> cut;
c5 1b    vector<int> vis(g.g.size(), 0), st = {s};
6f 32    vis[s] = 1;
ad 3c    while (st.size()) {
14 b1        int u = st.back(); st.pop_back();
54 32        for (auto e : g.g[u]) if (!vis[e.to] and e.flow < e.cap)
b3 c1            vis[e.to] = 1, st.push_back(e.to);
0a cb    }
3c 48    for (int i = 0; i < g.g.size(); i++) for (auto e : g.g[i])
88 9d        if (vis[i] and !vis[e.to] and !e.res) cut.emplace_back(i,
e.to);
e4 d1    return cut;
d9 cb }

```

## 7.12 Dominator Tree - Kawakami

```

// Se vira pra usar ai
//
// build - O(m log(n))
// dominates - O(1)
// c80920

1a 1a    int n;

14 bb    namespace d_tree {
45 04        vector<int> g[MAX];

// The dominator tree
9c b3        vector<int> tree[MAX];
bd 5a        int dfs_l[MAX], dfs_r[MAX];

// Auxiliary data
35 a2        vector<int> rg[MAX], bucket[MAX];
c8 3e        int idom[MAX], sdom[MAX], prv[MAX], pre[MAX];
0d 44        int ancestor[MAX], label[MAX];
c2 56        vector<int> preorder;

09 76    void dfs(int v) {
5c 6a        static int t = 0;
73 db        pre[v] = ++t;
1b 76        sdom[v] = label[v] = v;
eb a3        preorder.push_back(v);
c1 d0        for (int nxt: g[v]) {
7a 56            if (sdom[nxt] == -1) {
23 ee                prv[nxt] = v;
1e 90                dfs(nxt);
df cb        }

```

```

3f 2b         rg[nxt].push_back(v);
09 cb     }
23 cb }
a6 62 int eval(int v) {
ed c9     if (ancestor[v] == -1) return v;
32 a7     if (ancestor[ancestor[v]] == -1) return label[v];
09 f3     int u = eval(ancestor[v]);
de b4     if (pre[sdom[u]] < pre[sdom[label[v]]]) label[v] = u;
52 66     ancestor[v] = ancestor[u];
4d c2     return label[v];
35 cb }
57 4b void dfs2(int v) {
69 6a     static int t = 0;
74 33     dfs_l[v] = t++;
ed 5e     for (int nxt: tree[v]) dfs2(nxt);
dc 8e     dfs_r[v] = t++;
08 cb }
db c2 void build(int s) {
14 60     for (int i = 0; i < n; i++) {
78 e6         sdom[i] = pre[i] = ancestor[i] = -1;
6f 2e         rg[i].clear();
76 50         tree[i].clear();
01 66         bucket[i].clear();
2c cb     }
18 77     preorder.clear();
86 c6     dfs(s);
f3 12     if (preorder.size() == 1) return;
7d 3c     for (int i = preorder.size() - 1; i >= 1; i--) {
43 6c         int w = preorder[i];
58 a5         for (int v: rg[w]) {
62 5c             int u = eval(v);
6c a1             if (pre[sdom[u]] < pre[sdom[w]]) sdom[w] = sdom[u];
5f cb         }
ac 68         bucket[sdom[w]].push_back(w);
3c ea         ancestor[w] = prv[w];
b4 b9         for (int v: bucket[prv[w]]) {
9e 5c             int u = eval(v);
c4 97             idom[v] = (u == v) ? sdom[v] : u;
0c cb         }
5c 2c         bucket[prv[w]].clear();
13 cb     }
1d d0     for (int i = 1; i < preorder.size(); i++) {
55 6c         int w = preorder[i];
cf 14         if (idom[w] != sdom[w]) idom[w] = idom[idom[w]];
d0 32         tree[idom[w]].push_back(w);
64 cb     }
a4 8a     idom[s] = sdom[s] = -1;

```

```

40 1b         dfs2(s);
28 cb     }

        // Whether every path from s to v passes through u
0e 49     bool dominates(int u, int v) {
d5 c7         if (pre[v] == -1) return 1; // vacuously true
1f 2e         return dfs_l[u] <= dfs_l[v] && dfs_r[v] <= dfs_r[u];
28 cb     }
c8 21 };

```

## 7.13 Euler Path / Euler Cycle

```

// Para declarar: 'euler<true> E(n);' se quiser
// direcionado e com 'n' vertices
// As funcoes retornam um par com um booleano
// indicando se possui o cycle/path que voce pediu,
// e um vector de {vertice, id da aresta para chegar no vertice}
// Se for get_path, na primeira posicao o id vai ser -1
// get_path(src) tenta achar um caminho ou ciclo euleriano
// começando no vertice 'src'.
// Se achar um ciclo, o primeiro e ultimo vertice serao 'src'.
// Se for um P3, um possiveo retorno seria [0, 1, 2, 0]
// get_cycle() acha um ciclo euleriano se o grafo for euleriano.
// Se for um P3, um possivel retorno seria [0, 1, 2]
// (vertie inicial nao repete)
//
// 0(n+m)
// 7113df

63 63 template<bool directed=false> struct euler {
c0 1a     int n;
9e 4c     vector<vector<pair<int, int>>> g;
ef d6     vector<int> used;

f6 30     euler(int n_) : n(n_), g(n) {}
95 50     void add(int a, int b) {
b6 4c         int at = used.size();
fa c5         used.push_back(0);
f7 74         g[a].emplace_back(b, at);
75 fa         if (!directed) g[b].emplace_back(a, at);
5c cb     }
5c d4 #warning chamar para o src certo!
62 ee     pair<bool, vector<pair<int, int>>> get_path(int src) {
ab ba         if (!used.size()) return {true, {}};
e1 b2         vector<int> beg(n, 0);
91 4e         for (int& i : used) i = 0;

        // {{vertice, anterior}, label}

```

```

6b 36      vector<pair<pair<int, int>, int>> ret, st = {{{src, -1},
-1}}};
30 3c      while (st.size()) {
37 8f          int at = st.back().first.first;
21 00          int& it = beg[at];
a4 8a          while (it < g[at].size() and used[g[at][it].second])
            it++;
90 8e          if (it == g[at].size()) {
33 9d              if (ret.size() and ret.back().first.second != at)
08 b8                  return {false, {}};
19 42              ret.push_back(st.back()), st.pop_back();
2d 9d          } else {
89 da              st.push_back({g[at][it].first, at},
g[at][it].second);
4d eb              used[g[at][it].second] = 1;
ba cb          }
55 cb      }
41 a1      if (ret.size() != used.size()+1) return {false, {}};
8d f7      vector<pair<int, int>> ans;
da fd      for (auto i : ret) ans.emplace_back(i.first.first,
i.second);
22 45      reverse(ans.begin(), ans.end());
58 99      return {true, ans};
fa cb  }
91 9b      pair<bool, vector<pair<int, int>>> get_cycle() {
1c ba          if (!used.size()) return {true, {}};
a9 ad          int src = 0;
a0 34          while (!g[src].size()) src++;
84 68          auto ans = get_path(src);
5c 33          if (!ans.first or ans.second[0].first !=
ans.second.back().first)
7a b8              return {false, {}};
c7 35          ans.second[0].second = ans.second.back().second;
18 8b          ans.second.pop_back();
7f ba          return ans;
cd cb  }
71 21 };

```

## 7.14 Euler Tour Tree

```

// Mantem uma floresta enraizada dinamicamente
// e permite queries/updates em sub-arvore
//
// Chamar ETT E(n, v), passando n = numero de vertices
// e v = vector com os valores de cada vertice (se for vazio,
// constroi tudo com 0
//

```

```

// link(v, u) cria uma aresta de v pra u, de forma que u se torna
// o pai de v (eh preciso que v seja raiz anteriormente)
// cut(v) corta a resta de v para o pai
// query(v) retorna a soma dos valores da sub-arvore de v
// update(v, val) soma val em todos os vertices da sub-arvore de v
// update_v(v, val) muda o valor do vertice v para val
// is_in_subtree(v, u) responde se o vertice u esta na sub-arvore de v
//
// Tudo O(log(n)) com alta probabilidade
// c97d63

87 87 mt19937 rng((int)
chrono::steady_clock::now().time_since_epoch().count());

86 9f template<typename T> struct ETT {
    // treap
47 3c      struct node {
d2 ed          node *l, *r, *p;
a0 fa          int pr, sz;
e6 87          T val, sub, lazy;
e9 53          int id;
7a ff          bool f; // se eh o 'first'
d2 5e          int qt_f; // numero de firsts na subarvore
ce 7a          node(int id_, T v, bool f_ = 0) : l(NULL), r(NULL),
p(NULL), pr(rng()),
0f 62              sz(1), val(v), sub(v), lazy(), id(id_), f(f_),
qt_f(f_) {}
a1 a9          void prop() {
c4 d0              if (lazy != T()) {
29 02                  if (f) val += lazy;
1c 97                  sub += lazy*sz;
a9 b8                  if (l) l->lazy += lazy;
cc d3                  if (r) r->lazy += lazy;
0d cb              }
55 bf              lazy = T();
c6 cb          }
09 01          void update() {
12 8d              sz = 1, sub = val, qt_f = f;
2b 17              if (l) l->prop(), sz += l->sz, sub += l->sub, qt_f +=
l->qt_f;
97 11              if (r) r->prop(), sz += r->sz, sub += r->sub, qt_f +=
r->qt_f;
47 cb          }
28 21      };

dc bb      node* root;

```



```

26 73 int size(node* x) { return x ? x->sz : 0; }
b3 bc void join(node* l, node* r, node*& i) { // assume que l < r
95 98     if (!l or !r) return void(i = l ? l : r);
08 16     l->prop(), r->prop();
b1 ff     if (l->pr > r->pr) join(l->r, r, l->r), l->r->p = i = l;
68 98     else join(l, r->l, r->l), r->l->p = i = r;
ab bd     i->update();
b5 cb }
e5 a2 void split(node* i, node*& l, node*& r, int v, int key = 0) {
31 26     if (!i) return void(r = l = NULL);
84 c8     i->prop();
8e d9     if (key + size(i->l) < v) {
85 44         split(i->r, i->r, r, v, key+size(i->l)+1), l = i;
a0 a2         if (r) r->p = NULL;
30 6e         if (i->r) i->r->p = i;
9c 9d     } else {
b8 98         split(i->l, l, i->l, v, key), r = i;
fc 5a         if (l) l->p = NULL;
89 89         if (i->l) i->l->p = i;
76 cb     }
9e bd     i->update();
79 cb }
cc ac int get_idx(node* i) {
7b 6c     int ret = size(i->l);
64 48     for (; i->p; i = i->p) {
ad fb         node* pai = i->p;
d8 8a         if (i != pai->l) ret += size(pai->l) + 1;
4c cb     }
3f ed     return ret;
d4 cb }
57 04 node* get_min(node* i) {
7c 43     if (!i) return NULL;
a8 f8     return i->l ? get_min(i->l) : i;
95 cb }
0b f0 node* get_max(node* i) {
1c 43     if (!i) return NULL;
f3 42     return i->r ? get_max(i->r) : i;
14 cb }
// fim da treap

d8 4f vector<node*> first, last;

93 f8 ETT(int n, vector<T> v = {}) : root(NULL), first(n), last(n) {
a6 c5     if (!v.size()) v = vector<T>(n);
09 60     for (int i = 0; i < n; i++) {
cd a0         first[i] = last[i] = new node(i, v[i], 1);
8e 46         join(root, first[i], root);

```

```

68 cb     }
ba cb }
d0 83 ETT(const ETT& t) { throw logic_error("Nao copiar a ETT!"); }
ab c0 ~ETT() {
5a 60     vector<node*> q = {root};
92 40     while (q.size()) {
5b e5         node* x = q.back(); q.pop_back();
5d ee         if (!x) continue;
77 1c         q.push_back(x->l), q.push_back(x->r);
87 bf         delete x;
49 cb     }
ff cb }

67 15 pair<int, int> get_range(int i) {
c4 67     return {get_idx(first[i]), get_idx(last[i])};
93 cb }
1e 7a void link(int v, int u) { // 'v' tem que ser raiz
ba 89     auto [lv, rv] = get_range(v);
e2 f1     int ru = get_idx(last[u]);

91 4b     node* V;
85 df     node *L, *M, *R;
21 11     split(root, M, R, rv+1), split(M, L, M, lv);
2f f1     V = M;
f8 a2     join(L, R, root);

7b e6     split(root, L, R, ru+1);
79 36     join(L, V, L);
b6 7e     join(L, last[u] = new node(u, T() /* elemento neutro */),
L);
7c a2     join(L, R, root);
19 cb }
a7 4e void cut(int v) {
b1 89     auto [l, r] = get_range(v);

b9 df     node *L, *M, *R;
88 dc     split(root, M, R, r+1), split(M, L, M, l);
91 de     node *LL = get_max(L), *RR = get_min(R);
88 71     if (LL and RR and LL->id == RR->id) { // remove duplicata
42 e8         if (last[RR->id] == RR) last[RR->id] = LL;
38 99         node *A, *B;
0c 6b         split(R, A, B, 1);
1f 10         delete A;
da 9d         R = B;
2c cb     }
b4 a2     join(L, R, root);
35 a0     join(root, M, root);

```

```

fb cb    }
85 80    T query(int v) {
02 89        auto [l, r] = get_range(v);
51 df        node *L, *M, *R;
53 dc        split(root, M, R, r+1), split(M, L, M, l);
4c d4        T ans = M->sub;
8e 69        join(L, M, M), join(M, R, root);
2b ba        return ans;
ae cb    }
ff 93    void update(int v, T val) { // soma val em todo mundo da
        subarvore
b5 89        auto [l, r] = get_range(v);
e4 df        node *L, *M, *R;
70 dc        split(root, M, R, r+1), split(M, L, M, l);
ca 40        M->lazy += val;
06 69        join(L, M, M), join(M, R, root);
3a cb    }
6b 12    void update_v(int v, T val) { // muda o valor de v pra val
bf ac        int l = get_idx(first[v]);
3a df        node *L, *M, *R;
c0 d0        split(root, M, R, l+1), split(M, L, M, l);
1c 25        M->val = M->sub = val;
a8 69        join(L, M, M), join(M, R, root);
b4 cb    }
50 93    bool is_in_subtree(int v, int u) { // se u ta na subtree de v
8f 89        auto [lv, rv] = get_range(v);
f0 6e        auto [lu, ru] = get_range(u);
a6 73        return lv <= lu and ru <= rv;
7f cb    }

2d 35    void print(node* i) {
56 ea        if (!i) return;
91 a1        print(i->l);
ef 74        cout << i->id+1 << " ";
09 f1        print(i->r);
61 cb    }
10 06    void print() { print(root); cout << endl; }
c9 21 };

```

## 7.15 Floyd-Warshall

```

// encontra o menor caminho entre todo
// par de vertices e detecta ciclo negativo
// retorna 1 sse ha ciclo negativo
// d[i][i] deve ser 0
// para i != j, d[i][j] deve ser w se ha uma aresta
// (i, j) de peso w, INF caso contrario

```

```

//
// 0(n^3)
// ea05be

1a 1a int n;
1c ae int d[MAX][MAX];

9b 73 bool floyd_warshall() {
e8 e2     for (int k = 0; k < n; k++)
9e 83     for (int i = 0; i < n; i++)
d0 f9     for (int j = 0; j < n; j++)
87 0a         d[i][j] = min(d[i][j], d[i][k] + d[k][j]);

32 83     for (int i = 0; i < n; i++)
0c 75         if (d[i][i] < 0) return 1;

1f bb     return 0;
ea cb }

```

## 7.16 Functional Graph

```

// rt[i] fala o ID da raiz associada ao vertice i
// d[i] fala a profundidade (0 sse ta no ciclo)
// pos[i] fala a posicao de i no array que eh a concat. dos ciclos
// build(f, val) recebe a funcao f e o custo de ir de
// i para f[i] (por default, val = f)
// f_k(i, k) fala onde i vai parar se seguir k arestas
// path(i, k) fala o custo (soma) seguir k arestas a partir de i
// Se quiser outra operacao, da pra alterar facil o codigo
//Codigo um pouco louco, tenho que admitir
//
// build - 0(n)
// f_k    - 0(log(min(n, k)))
// path   - 0(log(min(n, k)))
// 5ifabe

6e 6e namespace func_graph {
e8 1a     int n;
eb ce     int f[MAX], vis[MAX], d[MAX];
18 f8     int p[MAX], pp[MAX], rt[MAX], pos[MAX];
17 eb     int sz[MAX], comp;
d4 6a     vector<vector<int>> ciclo;
cd 40     ll val[MAX], jmp[MAX], seg[2*MAX];

31 97     ll op(ll a, ll b) { return a+b; }; // mudar a operacao aqui
4b 27     void dfs(int i, int t = 2) {
ef 9c         vis[i] = t;

```

```

cc f0      if (vis[f[i]] >= 2) { // começa ciclo - f[i] eh o rep.
a9 e0      d[i] = 0, rt[i] = comp;
0e 74      sz[comp] = t - vis[f[i]] + 1;
16 97      p[i] = pp[i] = i, jmp[i] = val[i];
1f 15      ciclo.emplace_back();
4c bf      ciclo.back().push_back(i);
33 9d      } else {
43 c1      if (!vis[f[i]]) dfs(f[i], t+1);
61 8c      rt[i] = rt[f[i]];
eb 19      if (sz[comp]+1) { // to no ciclo
34 d0      d[i] = 0;
9b 97      p[i] = pp[i] = i, jmp[i] = val[i];
ab bf      ciclo.back().push_back(i);
38 9d      } else { // nao to no ciclo
e0 00      d[i] = d[f[i]]+1, p[i] = f[i];
fd 51      pp[i] = 2*d[pp[f[i]]] == d[pp[pp[f[i]]]]+d[f[i]] ?
pp[pp[f[i]]] : f[i];
ce 11      jmp[i] = pp[i] == f[i] ? val[i] : op(val[i],
op(jmp[f[i]], jmp[pp[f[i]]]));
6a cb      }
f3 cb      }
9f e4      if (f[ciclo[rt[i]][0]] == i) comp++; // fim do ciclo
0a 29      vis[i] = 1;
a3 cb      }
a6 1d      void build(vector<int> f_, vector<int> val_ = {}) {
69 bc      n = f_.size(), comp = 0;
99 52      if (!val_.size()) val_ = f_;
32 83      for (int i = 0; i < n; i++)
7d 99      f[i] = f_[i], val[i] = val_[i], vis[i] = 0, sz[i] = -1;

ab e7      ciclo.clear();
6d 15      for (int i = 0; i < n; i++) if (!vis[i]) dfs(i);
5c 6b      int t = 0;
e1 da      for (auto& c : ciclo) {
c8 33      reverse(c.begin(), c.end());
d5 ea      for (int j : c) {
4b 85      pos[j] = t;
7e 94      seg[n+t] = val[j];
2c c8      t++;
fa cb      }
d5 cb      }
d4 dc      for (int i = n-1; i; i--) seg[i] = op(seg[2*i],
seg[2*i+1]);
94 cb      }

07 28      int f_k(int i, ll k) {
49 1b      while (d[i] and k) {

```

```

7e 77      int big = d[i] - d[pp[i]];
fe de      if (big <= k) k -= big, i = pp[i];
f4 58      else k--, i = p[i];
57 cb      }
10 77      if (!k) return i;
27 a1      return ciclo[rt[i]][(pos[i] - pos[ciclo[rt[i]][0]] + k) %
sz[rt[i]]];
f2 cb      }
fb 04      ll path(int i, ll k) {
85 3c      auto query = [&](int l, int r) {
ea 3e      ll q = 0;
66 47      for (l += n, r += n; l <= r; ++l/=2, --r/=2) {
ac 27      if (l%2 == 1) q = op(q, seg[l]);
50 1f      if (r%2 == 0) q = op(q, seg[r]);
2b cb      }
54 be      return q;
70 21      };
fb b7      ll ret = 0;
3e 1b      while (d[i] and k) {
01 77      int big = d[i] - d[pp[i]];
f0 32      if (big <= k) k -= big, ret = op(ret, jmp[i]), i =
pp[i];
19 f9      else k--, ret = op(ret, val[i]), i = p[i];
8f cb      }
4c e3      if (!k) return ret;
e1 a9      int first = pos[ciclo[rt[i]][0]], last =
pos[ciclo[rt[i]].back()];

// k/sz[rt[i]] voltas completas
b3 43      if (k/sz[rt[i]]) ret = op(ret, k/sz[rt[i]] * query(first,
last));

ab 9a      k %= sz[rt[i]];
25 e3      if (!k) return ret;
47 8e      int l = pos[i], r = first + (pos[i] - first + k - 1) %
sz[rt[i]];
c2 98      if (l <= r) return op(ret, query(l, r));
b5 68      return op(ret, op(query(l, last), query(first, r)));
78 cb      }
51 cb      }

```

## 7.17 Heavy-Light Decomposition - aresta

```

// SegTree de soma
// query / update de soma das arestas
//
// Complexidades:

```

```

// build - O(n)
// query_path - O(log^2 (n))
// update_path - O(log^2 (n))
// query_subtree - O(log(n))
// update_subtree - O(log(n))

// namespace seg { ... }

// 599946
82 82 namespace hld {
35 c0     vector<pair<int, int> > g[MAX];
a7 e6     int pos[MAX], sz[MAX];
59 7c     int sobe[MAX], pai[MAX];
e5 09     int h[MAX], v[MAX], t;

8a 0c     void build_hld(int k, int p = -1, int f = 1) {
0f 18         v[pos[k] = t++] = sobe[k]; sz[k] = 1;
ed 41         for (auto& i : g[k]) if (i.first != p) {
9c dd             auto [u, w] = i;
65 a7             sobe[u] = w; pai[u] = k;
b6 0c             h[u] = (i == g[k][0] ? h[k] : u);
24 da             build_hld(u, k, f); sz[k] += sz[u];

2a 86             if (sz[u] > sz[g[k][0].first] or g[k][0].first == p)
38 9a                 swap(i, g[k][0]);
0f cb         }
f3 66         if (p*f == -1) build_hld(h[k] = k, -1, t = 0);
99 cb     }
b3 1f     void build(int root = 0) {
a6 a3         t = 0;
e6 29         build_hld(root);
40 c8         seg::build(t, v);
5c cb     }
a3 3f     ll query_path(int a, int b) {
10 2d         if (a == b) return 0;
e2 aa         if (pos[a] < pos[b]) swap(a, b);

17 29         if (h[a] == h[b]) return seg::query(pos[b]+1, pos[a]);
14 fc         return seg::query(pos[h[a]], pos[a]) +
            query_path(pai[h[a]], b);
da cb     }
45 92     void update_path(int a, int b, int x) {
1e d5         if (a == b) return;
f3 aa         if (pos[a] < pos[b]) swap(a, b);

58 88         if (h[a] == h[b]) return (void)seg::update(pos[b]+1,
            pos[a], x);

```

```

55 70         seg::update(pos[h[a]], pos[a], x); update_path(pai[h[a]],
            b, x);
3b cb     }
5e d0     ll query_subtree(int a) {
31 b9         if (sz[a] == 1) return 0;
22 2f         return seg::query(pos[a]+1, pos[a]+sz[a]-1);
90 cb     }
5f ac     void update_subtree(int a, int x) {
c6 a5         if (sz[a] == 1) return;
e6 9c         seg::update(pos[a]+1, pos[a]+sz[a]-1, x);
9b cb     }
e7 7b     int lca(int a, int b) {
f9 aa         if (pos[a] < pos[b]) swap(a, b);
82 ca         return h[a] == h[b] ? b : lca(pai[h[a]], b);
2c cb     }
59 cb }

```

## 7.18 Heavy-Light Decomposition - vertice

```

// SegTree de soma
// query / update de soma dos vertices
//
// Complexidades:
// build - O(n)
// query_path - O(log^2 (n))
// update_path - O(log^2 (n))
// query_subtree - O(log(n))
// update_subtree - O(log(n))

// namespace seg { ... }

// de3d84
82 82 namespace hld {
12 04     vector<int> g[MAX];
52 e6     int pos[MAX], sz[MAX];
e2 bd     int peso[MAX], pai[MAX];
91 09     int h[MAX], v[MAX], t;

4b 0c     void build_hld(int k, int p = -1, int f = 1) {
ed b1         v[pos[k] = t++] = peso[k]; sz[k] = 1;
62 b9         for (auto& i : g[k]) if (i != p) {
ed 78             pai[i] = k;
c3 26             h[i] = (i == g[k][0] ? h[k] : i);
b5 19             build_hld(i, k, f); sz[k] += sz[i];

7b cd             if (sz[i] > sz[g[k][0]] or g[k][0] == p) swap(i,
                g[k][0]);

```

```

33 cb    }
1b 66    if (p*f == -1) build_hld(h[k] = k, -1, t = 0);
ac cb    }
b3 1f    void build(int root = 0) {
94 a3        t = 0;
20 29        build_hld(root);
bd c8        seg::build(t, v);
dd cb    }
69 3f    ll query_path(int a, int b) {
d0 aa        if (pos[a] < pos[b]) swap(a, b);

c0 4b        if (h[a] == h[b]) return seg::query(pos[b], pos[a]);
27 fc        return seg::query(pos[h[a]], pos[a]) +
        query_path(pai[h[a]], b);
56 cb    }
c6 92    void update_path(int a, int b, int x) {
57 aa        if (pos[a] < pos[b]) swap(a, b);

2a 19        if (h[a] == h[b]) return (void)seg::update(pos[b], pos[a],
        x);
7f 70        seg::update(pos[h[a]], pos[a], x); update_path(pai[h[a]],
        b, x);
1c cb    }
d4 d0    ll query_subtree(int a) {
67 b3        return seg::query(pos[a], pos[a]+sz[a]-1);
9f cb    }
4c ac    void update_subtree(int a, int x) {
ce a2        seg::update(pos[a], pos[a]+sz[a]-1, x);
15 cb    }
00 7b    int lca(int a, int b) {
d1 aa        if (pos[a] < pos[b]) swap(a, b);
c2 ca        return h[a] == h[b] ? b : lca(pai[h[a]], b);
db cb    }
de cb    }

```

## 7.19 Heavy-Light Decomposition sem Update

```

// query de min do caminho
//
// Complexidades:
// build - O(n)
// query_path - O(log(n))
// ee6991

```

```

82 82 namespace hld {
35 c0    vector<pair<int, int> > g[MAX];
a7 e6    int pos[MAX], sz[MAX];

```

```

59 7c    int sobe[MAX], pai[MAX];
e5 09    int h[MAX], v[MAX], t;
f1 ea    int men[MAX], seg[2*MAX];

d7 0c    void build_hld(int k, int p = -1, int f = 1) {
52 18        v[pos[k] = t++] = sobe[k]; sz[k] = 1;
1f 41        for (auto& i : g[k]) if (i.first != p) {
1d 1f            sobe[i.first] = i.second; pai[i.first] = k;
aa 6f            h[i.first] = (i == g[k][0] ? h[k] : i.first);
91 87            men[i.first] = (i == g[k][0] ? min(men[k], i.second) :
        i.second);
03 4b            build_hld(i.first, k, f); sz[k] += sz[i.first];

f7 bc            if (sz[i.first] > sz[g[k][0].first] or g[k][0].first
        == p)
a0 9a                swap(i, g[k][0]);
c4 cb        }
0d 66        if (p*f == -1) build_hld(h[k] = k, -1, t = 0);
17 cb    }
01 1f    void build(int root = 0) {
8c a3        t = 0;
39 29        build_hld(root);
ce 3a        for (int i = 0; i < t; i++) seg[i+t] = v[i];
f8 8d        for (int i = t-1; i; i--) seg[i] = min(seg[2*i],
        seg[2*i+1]);
81 cb    }
a8 f0    int query_path(int a, int b) {
24 49        if (a == b) return INF;
8b aa        if (pos[a] < pos[b]) swap(a, b);

6d 98        if (h[a] != h[b]) return min(men[a], query_path(pai[h[a]],
        b));
4d 46        int ans = INF, x = pos[b]+1+t, y = pos[a]+t;
27 64        for (; x <= y; ++x/=2, --y/=2) ans = min({ans, seg[x],
        seg[y]});
68 ba        return ans;
11 cb    }
ee 21    };

```

## 7.20 Isomorfismo de arvores

```

// thash() retorna o hash da arvore (usando centroids como vertices
        especiais).
// Duas arvores sao isomorfas sse seu hash eh o mesmo
//
// O(|V|.log(|V|))
// 8fb6bb

```

```

91 91 map<vector<int>, int> mphash;

fa df struct tree {
a8 1a     int n;
2c 78     vector<vector<int>> g;
a8 34     vector<int> sz, cs;

7d 1b     tree(int n_) : n(n_), g(n_), sz(n_) {}

b7 76     void dfs_centroid(int v, int p) {
91 58         sz[v] = 1;
43 fa         bool cent = true;
57 18         for (int u : g[v]) if (u != p) {
73 36             dfs_centroid(u, v), sz[v] += sz[u];
0a e9             if(sz[u] > n/2) cent = false;
79 cb         }
2c 1f         if (cent and n - sz[v] <= n/2) cs.push_back(v);
84 cb     }
b3 78     int fhash(int v, int p) {
e8 54         vector<int> h;
73 33         for (int u : g[v]) if (u != p) h.push_back(fhash(u, v));
f9 1c         sort(h.begin(), h.end());
cb 3a         if (!mphash.count(h)) mphash[h] = mphash.size();
ee bb         return mphash[h];
4a cb     }
9c 38     ll thash() {
cb 23         cs.clear();
0a 3a         dfs_centroid(0, -1);
8d 16         if (cs.size() == 1) return fhash(cs[0], -1);
48 77         ll h1 = fhash(cs[0], cs[1]), h2 = fhash(cs[1], cs[0]);
2a fa         return (min(h1, h2) << 30) + max(h1, h2);
6f cb     }
8f 21 };

```

## 7.21 Kosaraju

```

// 0(n + m)
// a4f310

1a 1a int n;
a7 04 vector<int> g[MAX];
32 58 vector<int> gi[MAX]; // grafo invertido
c9 c5 int vis[MAX];
10 ee stack<int> S;
03 a5 int comp[MAX]; // componente conexo de cada vertice

```

```

a8 1c void dfs(int k) {
3c 59     vis[k] = 1;
aa 54     for (int i = 0; i < (int) g[k].size(); i++)
15 8d         if (!vis[g[k][i]]) dfs(g[k][i]);

b8 58     S.push(k);
08 cb }

d4 43 void scc(int k, int c) {
42 59     vis[k] = 1;
c6 52     comp[k] = c;
80 ff     for (int i = 0; i < (int) gi[k].size(); i++)
e9 bf         if (!vis[gi[k][i]]) scc(gi[k][i], c);
5d cb }

30 db void kosaraju() {
4f 99     for (int i = 0; i < n; i++) vis[i] = 0;
c3 15     for (int i = 0; i < n; i++) if (!vis[i]) dfs(i);

be 99     for (int i = 0; i < n; i++) vis[i] = 0;
22 d3     while (S.size()) {
7a 70         int u = S.top();
bf 7d         S.pop();
d6 f4         if (!vis[u]) scc(u, u);
5b cb     }
a4 cb }

```

## 7.22 Kruskal

```

// Gera e retorna uma AGM e seu custo total a partir do vetor de
// arestas (edg)
// do grafo
//
// 0(m log(m) + m a(m))
// 864875

1b 1b vector<tuple<int, int, int>> edg; // {peso, [x,y]}

// DSU em 0(a(n))
9d 4a void dsu_build();
86 d7 int find(int a);
d5 36 void unite(int a, int b);

64 c6 pair<ll, vector<tuple<int, int, int>>> kruskal(int n) {
bd 8d     dsu_build(n);
b6 e3     sort(edg.begin(), edg.end());

```

```

5c 85    ll cost = 0;
25 97    vector<tuple<int, int, int>> mst;
16 fe    for (auto [w,x,y] : edg) if (find(x) != find(y)) {
f2 9d        mst.emplace_back(w, x, y);
bc 45        cost += w;
32 05        unite(x,y);
eb cb    }
38 5d    return {cost, mst};
86 cb }

```

## 7.23 Kuhn

```

// Computa matching maximo em grafo bipartido
// 'n' e 'm' sao quantos vertices tem em cada particao
// chamar add(i, j) para add aresta entre o cara i
// da particao A, e o cara j da particao B
// (entao i < n, j < m)
// Para recuperar o matching, basta olhar 'ma' e 'mb'
// 'recover' recupera o min vertex cover como um par de
// {caras da particao A, caras da particao B}
//
// O(|V| * |E|)
// Na pratica, parece rodar tao rapido quanto o Dinic

87 87 mt19937 rng((int)
    chrono::steady_clock::now().time_since_epoch().count());

// b0dda3
0b 6c struct kuhn {
e2 14    int n, m;
a3 78    vector<vector<int>> g;
4f d3    vector<int> vis, ma, mb;

5e 40    kuhn(int n_, int m_) : n(n_), m(m_), g(n),
bc 8a        vis(n+m), ma(n, -1), mb(m, -1) {}

41 ba    void add(int a, int b) { g[a].push_back(b); }

66 ca    bool dfs(int i) {
14 29        vis[i] = 1;
24 29        for (int j : g[i]) if (!vis[n+j]) {
a1 8c            vis[n+j] = 1;
6e 2c            if (mb[j] == -1 or dfs(mb[j])) {
e6 bf                ma[i] = j, mb[j] = i;
69 8a                return true;
24 cb            }
13 cb        }

```

```

c8 d1        return false;
5f cb    }
be bf    int matching() {
26 1a        int ret = 0, aum = 1;
a4 5a        for (auto& i : g) shuffle(i.begin(), i.end(), rng);
30 39        while (aum) {
43 61            for (int j = 0; j < m; j++) vis[n+j] = 0;
cb c5            aum = 0;
a0 83            for (int i = 0; i < n; i++)
5b 01                if (ma[i] == -1 and dfs(i)) ret++, aum = 1;
31 cb        }
8a ed        return ret;
96 cb    }
c1 21 };

// 55fb67
c9 eb pair<vector<int>, vector<int>> recover(kuhn& K) {
16 e8    K.matching();
2a 50    int n = K.n, m = K.m;
6c 9d    for (int i = 0; i < n+m; i++) K.vis[i] = 0;
a0 bd    for (int i = 0; i < n; i++) if (K.ma[i] == -1) K.dfs(i);
b1 8a    vector<int> ca, cb;
be 57    for (int i = 0; i < n; i++) if (!K.vis[i]) ca.push_back(i);
97 f2    for (int i = 0; i < m; i++) if (K.vis[n+i]) cb.push_back(i);
d1 aa    return {ca, cb};
85 cb }

```

## 7.24 LCA com binary lifting

```

// Assume que um vertice eh ancestral dele mesmo, ou seja,
// se a eh ancestral de b, lca(a, b) = a
// MAX2 = ceil(log(MAX))
//
// Complexidades:
// build - O(n log(n))
// lca - O(log(n))
// b674ca

67 67 vector<vector<int> > g(MAX);
76 41 int n, p;
9a e7 int pai[MAX2][MAX];
3c 99 int in[MAX], out[MAX];

4a 1c void dfs(int k) {
07 fd    in[k] = p++;
d6 54    for (int i = 0; i < (int) g[k].size(); i++)
67 9b        if (in[g[k][i]] == -1) {

```

```

6b ba          pai[0][g[k][i]] = k;
29 c3          dfs(g[k][i]);
51 cb          }
10 26 out[k] = p++;
8a cb }

df c1 void build(int raiz) {
52 a6   for (int i = 0; i < n; i++) pai[0][i] = i;
14 c6   p = 0, memset(in, -1, sizeof in);
49 ec   dfs(raiz);

        // pd dos pais
35 51   for (int k = 1; k < MAX2; k++) for (int i = 0; i < n; i++)
61 d3       pai[k][i] = pai[k - 1][pai[k - 1][i]];
de cb }

da 00 bool anc(int a, int b) { // se a eh ancestral de b
a5 bf   return in[a] <= in[b] and out[a] >= out[b];
df cb }

9d 7b int lca(int a, int b) {
c5 86   if (anc(a, b)) return a;
f6 e5   if (anc(b, a)) return b;

        // sobe a
09 f7   for (int k = MAX2 - 1; k >= 0; k--)
d0 ac       if (!anc(pai[k][a], b)) a = pai[k][a];

fb 84   return pai[0][a];
b6 cb }

// Alternativamente:
// 'binary lifting' gastando O(n) de memoria
// Da pra add folhas e fazer queries online
// 3 vezes o tempo do binary lifting normal
//
// build - O(n)
// kth, lca, dist - O(log(n))
// 89a97a

bc 9c int d[MAX], p[MAX], pp[MAX];

48 d4 void set_root(int i) { p[i] = pp[i] = i, d[i] = 0; }

d1 e9 void add_leaf(int i, int u) {
89 e0   p[i] = u, d[i] = d[u]+1;
6f b1   pp[i] = 2*d[pp[u]] == d[pp[pp[u]]]+d[u] ? pp[pp[u]] : u;

```

```

3e cb }

7b c3 int kth(int i, int k) {
0a 4e   int dd = max(0, d[i]-k);
5d 93   while (d[i] > dd) i = d[pp[i]] >= dd ? pp[i] : p[i];
49 d9   return i;
e7 cb }

8d 7b int lca(int a, int b) {
05 a6   if (d[a] < d[b]) swap(a, b);
8d 6c   while (d[a] > d[b]) a = d[pp[a]] >= d[b] ? pp[a] : p[a];
33 98   while (a != b) {
f9 93       if (pp[a] != pp[b]) a = pp[a], b = pp[b];
1d e7       else a = p[a], b = p[b];
ae cb   }
a4 3f   return a;
be cb }

77 4f int dist(int a, int b) { return d[a]+d[b]-2*d[lca(a,b)]; }

03 04 vector<int> g[MAX];

7b 3a void build(int i, int pai=-1) {
bb 5c   if (pai == -1) set_root(i);
a2 15   for (int j : g[i]) if (j != pai) {
b1 d3       add_leaf(j, i);
9a b2       build(j, i);
15 cb   }
d6 cb }

7.25 LCA com HLD

// Assume que um vertice eh ancestral dele mesmo, ou seja,
// se a eh ancestral de b, lca(a, b) = a
// Para buildar pasta chamar build(root)
// anc(a, b) responde se 'a' eh ancestral de 'b'
//
// Complexidades:
// build - O(n)
// lca - O(log(n))
// anc - O(1)
// fb22c1

04 04 vector<int> g[MAX];
a9 71 int pos[MAX], h[MAX], sz[MAX];
02 ff int pai[MAX], t;

```



```

1b 8b void build(int k, int p = -1, int f = 1) {
53 bc     pos[k] = t++; sz[k] = 1;
70 e2     for (int& i : g[k]) if (i != p) {
cd 78         pai[i] = k;
14 26         h[i] = (i == g[k][0] ? h[k] : i);
b4 cb         build(i, k, f); sz[k] += sz[i];

d8 cd         if (sz[i] > sz[g[k][0]] or g[k][0] == p) swap(i, g[k][0]);
30 cb     }
1b 3d     if (p*f == -1) t = 0, h[k] = k, build(k, -1, 0);
60 cb }

80 7b int lca(int a, int b) {
26 aa     if (pos[a] < pos[b]) swap(a, b);
79 ca     return h[a] == h[b] ? b : lca(pai[h[a]], b);
a9 cb }

b7 00 bool anc(int a, int b) {
a8 db     return pos[a] <= pos[b] and pos[b] <= pos[a]+sz[a]-1;
fb cb }

```

## 7.26 LCA com RMQ

```

// Assume que um vertice eh ancestral dele mesmo, ou seja,
// se a eh ancestral de b, lca(a, b) = a
// dist(a, b) retorna a distancia entre a e b
//
// Complexidades:
// build - O(n)
// lca - O(1)
// dist - O(1)
// 22cde8 - rmq + lca

// 0214e8
1a 1a template<typename T> struct rmq {
9e 51     vector<T> v;
4b fc     int n; static const int b = 30;
52 70     vector<int> mask, t;

34 18     int op(int x, int y) { return v[x] < v[y] ? x : y; }
d6 ee     int msb(int x) { return __builtin_clz(1)-__builtin_clz(x); }
74 6a     rmq() {}
d5 43     rmq(const vector<T>& v_) : v(v_), n(v.size()), mask(n), t(n) {
2e 2e         for (int i = 0, at = 0; i < n; mask[i++] = at |= 1) {
2f a6             at = (at<<1)&((1<<b)-1);
af 76             while (at and op(i, i-msb(at&-at)) == i) at ^= at&-at;
26 cb         }

```

```

92 24         for (int i = 0; i < n/b; i++) t[i] =
b*i+b-1-msb(mask[b*i+b-1]);
77 39         for (int j = 1; (1<<j) <= n/b; j++) for (int i = 0;
i+(1<<j) <= n/b; i++)
f7 ba             t[n/b*j+i] = op(t[n/b*(j-1)+i],
t[n/b*(j-1)+i+(1<<(j-1))]);
1a cb     }
e2 c9     int small(int r, int sz = b) { return
r-msb(mask[r]&((1<<sz)-1)); }
2b b7     T query(int l, int r) {
89 27         if (r-l+1 <= b) return small(r, r-l+1);
69 7b         int ans = op(small(l+b-1), small(r));
ef e8         int x = l/b+1, y = r/b-1;
1c e2         if (x <= y) {
48 a4             int j = msb(y-x+1);
97 00             ans = op(ans, op(t[n/b*j+x], t[n/b*j+y-(1<<j)+1]));
5b cb         }
bb ba         return ans;
68 cb     }
02 21 };

// 645120
04 06 namespace lca {
99 04     vector<int> g[MAX];
0d 8e     int v[2*MAX], pos[MAX], dep[2*MAX];
d1 8b     int t;
3f 2d     rmq<int> RMQ;

8e 4c     void dfs(int i, int d = 0, int p = -1) {
f4 c9         v[t] = i, pos[i] = t, dep[t++] = d;
16 ca         for (int j : g[i]) if (j != p) {
bc 8e             dfs(j, d+1, i);
ce cf             v[t] = i, dep[t++] = d;
81 cb         }
5f cb     }
f6 78     void build(int n, int root) {
7c a3         t = 0;
ec 14         dfs(root);
64 3f         RMQ = rmq<int>(vector<int>(dep, dep+2*n-1));
52 cb     }
ab 7b     int lca(int a, int b) {
17 ab         a = pos[a], b = pos[b];
be 9c         return v[RMQ.query(min(a, b), max(a, b))];
b3 cb     }
f0 b5     int dist(int a, int b) {
94 67         return dep[pos[a]] + dep[pos[b]] - 2*dep[pos[lca(a, b)]];
2b cb     }

```

```
22 cb }
```

## 7.27 Line Tree

```
// Reduz min-query em arvore para RMQ
// Se o grafo nao for uma arvore, as queries
// sao sobre a arvore geradora maxima
// Queries de minimo
//
// build - O(n log(n))
// query - O(log(n))
// b1f418

1a 1a int n;

8f 3a namespace linetree {
63 f3     int id[MAX], seg[2*MAX], pos[MAX];
b2 43     vector<int> v[MAX], val[MAX];
6c 43     vector<pair<int, pair<int, int> > > ar;

4d dc     void add(int a, int b, int p) { ar.push_back({p, {a, b}}); }
7c 0a     void build() {
fd b0         sort(ar.rbegin(), ar.rend());
7a 0e         for (int i = 0; i < n; i++) id[i] = i, v[i] = {i},
            val[i].clear();
8d 8b         for (auto i : ar) {
68 c9             int a = id[i.second.first], b = id[i.second.second];
51 f6             if (a == b) continue;
25 c5             if (v[a].size() < v[b].size()) swap(a, b);
91 fb             for (auto j : v[b]) id[j] = a, v[a].push_back(j);
03 48             val[a].push_back(i.first);
fd 78             for (auto j : val[b]) val[a].push_back(j);
31 e3             v[b].clear(), val[b].clear();
20 cb         }
8c 8e         vector<int> vv;
ca 2c         for (int i = 0; i < n; i++) for (int j = 0; j <
            v[i].size(); j++) {
77 e5             pos[v[i][j]] = vv.size();
2f 94             if (j + 1 < v[i].size()) vv.push_back(val[i][j]);
56 1c             else vv.push_back(0);
98 cb         }
9d bb         for (int i = n; i < 2*n; i++) seg[i] = vv[i-n];
79 69         for (int i = n-1; i; i--) seg[i] = min(seg[2*i],
            seg[2*i+1]);
32 cb     }
9b 4e     int query(int a, int b) {
e1 59         if (id[a] != id[b]) return 0; // nao estao conectados
```

```
b1 ab         a = pos[a], b = pos[b];
68 d1         if (a > b) swap(a, b);
ef 19         b--;
76 38         int ans = INF;
6c 51         for (a += n, b += n; a <= b; ++a/=2, --b/=2) ans =
            min({ans, seg[a], seg[b]});
5f ba         return ans;
ae cb     }
b1 21 };
```

## 7.28 Link-cut Tree

```
// Link-cut tree padrao
//
// Todas as operacoes sao O(log(n)) amortizado
// e4e663

1e 1e namespace lct {
c5 3c     struct node {
8b 19         int p, ch[2];
0d 06         node() { p = ch[0] = ch[1] = -1; }
44 21     };

5d 5f     node t[MAX];

71 97     bool is_root(int x) {
a8 65         return t[x].p == -1 or (t[t[x].p].ch[0] != x and
            t[t[x].p].ch[1] != x);
e9 cb     }
a5 ed     void rotate(int x) {
ad 49         int p = t[x].p, pp = t[p].p;
a0 fc         if (!is_root(p)) t[pp].ch[t[pp].ch[1] == p] = x;
fc 25         bool d = t[p].ch[0] == x;
00 46         t[p].ch[!d] = t[x].ch[d], t[x].ch[d] = p;
5c a7         if (t[p].ch[!d]+1) t[t[p].ch[!d]].p = p;
9f 8f         t[x].p = pp, t[p].p = x;
7d cb     }
d7 07     void splay(int x) {
56 18         while (!is_root(x)) {
c6 49             int p = t[x].p, pp = t[p].p;
9f 0c             if (!is_root(pp)) rotate((t[pp].ch[0] == p)^(t[p].ch[0]
                == x) ? x : p);
a1 64             rotate(x);
d7 cb         }
dc cb     }
09 f1     int access(int v) {
79 0e         int last = -1;
```

```

ca 01     for (int w = v; w+1; last = w, splay(v), w = t[v].p)
7e 02         splay(w), t[w].ch[1] = (last == -1 ? -1 : v);
f0 3d     return last;
d3 cb    }
4a e8     int find_root(int v) {
3f 5e         access(v);
eb 3d         while (t[v].ch[0]+1) v = t[v].ch[0];
c3 f0         return splay(v), v;
55 cb    }
2b 14     void link(int v, int w) { // v deve ser raiz
5a 5e         access(v);
f6 10         t[v].p = w;
ed cb    }
41 4e     void cut(int v) { // remove aresta de v pro pai
91 5e         access(v);
67 26         t[v].ch[0] = t[t[v].ch[0]].p = -1;
c5 cb    }
03 bb     int lca(int v, int w) {
42 94         return access(v), access(w);
11 cb    }
e4 cb    }

```

## 7.29 Link-cut Tree - aresta

```

// Valores nas arestas
// rootify(v) torna v a raiz de sua arvore
// query(v, w) retorna a soma do caminho v--w
// update(v, w, x) soma x nas arestas do caminho v--w
//
// Todas as operacoes sao O(log(n)) amortizado
// 9ce48f

1e 1e namespace lct {
c5 3c     struct node {
8b 19         int p, ch[2];
1f 81         ll val, sub;
7c aa         bool rev;
c2 04         int sz, ar;
e8 4e         ll lazy;
b1 f9         node() {}
fc 7a         node(int v, int ar_) :
b4 54             p(-1), val(v), sub(v), rev(0), sz(ar_), ar(ar_), lazy(0) {
e7 b0             ch[0] = ch[1] = -1;
5f cb         }
82 21     };

84 c5     node t[2*MAX]; // MAXN + MAXQ

```

```

67 99     map<pair<int, int>, int> aresta;
c4 e4     int sz;

ed 95     void prop(int x) {
bc dc         if (t[x].lazy) {
a6 25             if (t[x].ar) t[x].val += t[x].lazy;
06 2a             t[x].sub += t[x].lazy*t[x].sz;
50 ed             if (t[x].ch[0]+1) t[t[x].ch[0]].lazy += t[x].lazy;
e3 94             if (t[x].ch[1]+1) t[t[x].ch[1]].lazy += t[x].lazy;
43 cb         }
7b aa         if (t[x].rev) {
fa f9             swap(t[x].ch[0], t[x].ch[1]);
79 37             if (t[x].ch[0]+1) t[t[x].ch[0]].rev ^= 1;
ff c3             if (t[x].ch[1]+1) t[t[x].ch[1]].rev ^= 1;
1c cb         }
a2 23         t[x].lazy = 0, t[x].rev = 0;
fd cb     }
e5 56     void update(int x) {
3c 1a         t[x].sz = t[x].ar, t[x].sub = t[x].val;
74 8c         for (int i = 0; i < 2; i++) if (t[x].ch[i]+1) {
8a 62             prop(t[x].ch[i]);
49 c4             t[x].sz += t[t[x].ch[i]].sz;
a7 26             t[x].sub += t[t[x].ch[i]].sub;
29 cb         }
d1 cb     }
73 97     bool is_root(int x) {
5f 65         return t[x].p == -1 or (t[t[x].p].ch[0] != x and
t[t[x].p].ch[1] != x);
42 cb     }
c3 ed     void rotate(int x) {
5d 49         int p = t[x].p, pp = t[p].p;
e5 fc         if (!is_root(p)) t[pp].ch[t[pp].ch[1] == p] = x;
82 25         bool d = t[p].ch[0] == x;
9f 46         t[p].ch[!d] = t[x].ch[d], t[x].ch[d] = p;
e4 a7         if (t[p].ch[!d]+1) t[t[p].ch[!d]].p = p;
8c 8f         t[x].p = pp, t[p].p = x;
d0 44         update(p), update(x);
b8 cb     }
9a 23     int splay(int x) {
62 18         while (!is_root(x)) {
49 49             int p = t[x].p, pp = t[p].p;
42 77             if (!is_root(p)) prop(pp);
2c be             prop(p), prop(x);
91 0c             if (!is_root(p)) rotate((t[pp].ch[0] == p)^(t[p].ch[0]
== x) ? x : p);
3b 64             rotate(x);
81 cb         }

```

```

bb aa      return prop(x), x;
f3 cb      }
b2 f1      int access(int v) {
e0 0e          int last = -1;
0c d9          for (int w = v; w+1; update(last = w), splay(v), w =
                t[v].p)
77 02              splay(w), t[w].ch[1] = (last == -1 ? -1 : v);
83 3d          return last;
9e cb      }
0b 9f      void make_tree(int v, int w=0, int ar=0) { t[v] = node(w, ar);
                }
0a e8      int find_root(int v) {
97 13          access(v), prop(v);
4a 9f          while (t[v].ch[0]+1) v = t[v].ch[0], prop(v);
3f 63          return splay(v);
85 cb      }
cc 82      bool conn(int v, int w) {
e7 2c          access(v), access(w);
17 b9          return v == w ? true : t[v].p != -1;
6b cb      }
5b 27      void rootify(int v) {
4e 5e          access(v);
cd a0          t[v].rev ^= 1;
b4 cb      }
d3 97      ll query(int v, int w) {
9d b5          rootify(w), access(v);
2c 24          return t[v].sub;
ab cb      }
2c 3f      void update(int v, int w, int x) {
e2 b5          rootify(w), access(v);
5e 12          t[v].lazy += x;
87 cb      }
cb 20      void link_(int v, int w) {
a5 82          rootify(w);
14 38          t[w].p = v;
cf cb      }
30 6b      void link(int v, int w, int x) { // v--w com peso x
23 37          int id = MAX + sz++;
47 11          aresta[make_pair(v, w)] = id;
05 a8          make_tree(id, x, 1);
38 c8          link_(v, id), link_(id, w);
27 cb      }
5a e6      void cut_(int v, int w) {
04 b5          rootify(w), access(v);
3c 26          t[v].ch[0] = t[t[v].ch[0]].p = -1;
68 cb      }
7c 03      void cut(int v, int w) {

```

```

3d b0          int id = aresta[make_pair(v, w)];
ae a4          cut_(v, id), cut_(id, w);
42 cb      }
06 bb      int lca(int v, int w) {
ac 5e          access(v);
9b a8          return access(w);
4b cb      }
9c cb      }

```

### 7.30 Link-cut Tree - vertice

```

// Valores nos vertices
// make_tree(v, w) cria uma nova arvore com um
// vertice soh com valor 'w'
// rootify(v) torna v a raiz de sua arvore
// query(v, w) retorna a soma do caminho v--w
// update(v, w, x) soma x nos vertices do caminho v--w
//
// Todas as operacoes sao O(log(n)) amortizado
// f9f489

1e 1e namespace lct {
c5 3c     struct node {
8b 19         int p, ch[2];
1f 81         ll val, sub;
7c aa         bool rev;
cd e4         int sz;
5b 4e         ll lazy;
11 f9         node() {}
b4 aa         node(int v) : p(-1), val(v), sub(v), rev(0), sz(1),
                lazy(0) {
d7 b0             ch[0] = ch[1] = -1;
a6 cb             }
5e 21         };

b9 5f         node t[MAX];

23 95         void prop(int x) {
50 dc             if (t[x].lazy) {
c5 9f                 t[x].val += t[x].lazy, t[x].sub += t[x].lazy*t[x].sz;
14 ed                 if (t[x].ch[0]+1) t[t[x].ch[0]].lazy += t[x].lazy;
b2 94                 if (t[x].ch[1]+1) t[t[x].ch[1]].lazy += t[x].lazy;
6e cb             }
da aa             if (t[x].rev) {
bd f9                 swap(t[x].ch[0], t[x].ch[1]);
07 37                 if (t[x].ch[0]+1) t[t[x].ch[0]].rev ^= 1;
56 c3                 if (t[x].ch[1]+1) t[t[x].ch[1]].rev ^= 1;

```

```

08 cb    }
56 23    t[x].lazy = 0, t[x].rev = 0;
c7 cb    }
fa 56    void update(int x) {
82 ec        t[x].sz = 1, t[x].sub = t[x].val;
75 8c        for (int i = 0; i < 2; i++) if (t[x].ch[i]+1) {
1f 62            prop(t[x].ch[i]);
fb c4            t[x].sz += t[t[x].ch[i]].sz;
f0 26            t[x].sub += t[t[x].ch[i]].sub;
58 cb        }
20 cb    }
44 97    bool is_root(int x) {
a0 65        return t[x].p == -1 or (t[t[x].p].ch[0] != x and
t[t[x].p].ch[1] != x);
8a cb    }
02 ed    void rotate(int x) {
e4 49        int p = t[x].p, pp = t[p].p;
bb fc        if (!is_root(pp)) t[pp].ch[t[pp].ch[1] == p] = x;
c7 25        bool d = t[p].ch[0] == x;
3f 46        t[p].ch[!d] = t[x].ch[d], t[x].ch[d] = p;
24 a7        if (t[p].ch[!d]+1) t[t[p].ch[!d]].p = p;
ca 8f        t[x].p = pp, t[p].p = x;
16 44        update(p), update(x);
92 cb    }
4b 23    int splay(int x) {
f7 18        while (!is_root(x)) {
93 49            int p = t[x].p, pp = t[p].p;
be 77            if (!is_root(p)) prop(pp);
91 be            prop(p), prop(x);
1e 0c            if (!is_root(p)) rotate((t[pp].ch[0] == p)^(t[p].ch[0]
== x) ? x : p);
bf 64            rotate(x);
f9 cb        }
c2 aa        return prop(x), x;
8a cb    }
88 f1    int access(int v) {
48 0e        int last = -1;
b6 d9        for (int w = v; w+1; update(last = w), splay(v), w =
t[v].p)
5d 02            splay(w), t[w].ch[1] = (last == -1 ? -1 : v);
5e 3d        return last;
5d cb    }
46 f1    void make_tree(int v, int w) { t[v] = node(w); }
aa e8    int find_root(int v) {
6b 13        access(v), prop(v);
b0 9f        while (t[v].ch[0]+1) v = t[v].ch[0], prop(v);
b6 63        return splay(v);

```

```

ee cb    }
3e f9    bool connected(int v, int w) {
2e 2c        access(v), access(w);
0f b9        return v == w ? true : t[v].p != -1;
37 cb    }
5e 27    void rootify(int v) {
08 5e        access(v);
b7 a0        t[v].rev ^= 1;
fd cb    }
81 97    ll query(int v, int w) {
fd b5        rootify(w), access(v);
b5 24        return t[v].sub;
cc cb    }
ee 3f    void update(int v, int w, int x) {
a3 b5        rootify(w), access(v);
77 12        t[v].lazy += x;
ef cb    }
57 14    void link(int v, int w) {
8d 82        rootify(w);
ba 38        t[w].p = v;
cb cb    }
e6 03    void cut(int v, int w) {
25 b5        rootify(w), access(v);
46 26        t[v].ch[0] = t[t[v].ch[0]].p = -1;
8b cb    }
69 bb    int lca(int v, int w) {
47 5e        access(v);
96 a8        return access(w);
da cb    }
f9 cb    }

```

### 7.31 Max flow com lower bound nas arestas

```

// add(a, b, l, r):
//  adiciona aresta de a pra b, onde precisa passar f de fluxo, l <= f
//  <= r
// add(a, b, c):
//  adiciona aresta de a pra b com capacidade c
//
// Mesma complexidade do Dinic
// 5f2379

91 91 struct lb_max_flow : dinic {
c4 5c    vector<int> d;
79 33    lb_max_flow(int n) : dinic(n + 2), d(n, 0) {}
b5 b1    void add(int a, int b, int l, int r) {
78 c9        d[a] -= l;

```

```

e3 f1      d[b] += 1;
33 01      dinic::add(a, b, r - 1);
c6 cb    }
7d 08 void add(int a, int b, int c) {
dc 10      dinic::add(a, b, c);
01 cb    }
49 7a bool has_circulation() {
39 50      int n = d.size();

39 85      ll cost = 0;
41 60      for (int i = 0; i < n; i++) {
d9 c6          if (d[i] > 0) {
e7 f5              cost += d[i];
d5 d0              dinic::add(n, i, d[i]);
59 9c          } else if (d[i] < 0) {
e9 76              dinic::add(i, n+1, -d[i]);
f9 cb          }
ae cb      }

82 28      return (dinic::max_flow(n, n+1) == cost);
98 cb    }
fb 7b bool has_flow(int src, int snk) {
d2 65      dinic::add(snk, src, INF);
7d e4      return has_circulation();
bb cb    }
2b 4e ll max_flow(int src, int snk) {
3a ee      if (!has_flow(src, snk)) return -1;
5d ea      dinic::F = 0;
cf 62      return dinic::max_flow(src, snk);
be cb    }
5f 21 };

```

## 7.32 MinCostMaxFlow

```

// min_cost_flow(s, t, f) computa o par (fluxo, custo)
// com max(fluxo) <= f que tenha min(custo)
// min_cost_flow(s, t) -> Fluxo maximo de custo minimo de s pra t
// Se for um dag, da pra substituir o SPFA por uma DP pra nao
// pagar O(nm) no comeco
// Se nao tiver aresta com custo negativo, nao precisa do SPFA
//
// O(nm + f * m log n)
// 697b4c

12 12 template<typename T> struct mcmf {
74 67     struct edge {
f5 b7         int to, rev, flow, cap; // para, id da reversa, fluxo,

```

```

        capacidade
70 7f         bool res; // se eh reversa
59 63         T cost; // custo da unidade de fluxo
be 89         edge() : to(0), rev(0), flow(0), cap(0), cost(0),
            res(false) {}
1f 1d         edge(int to_, int rev_, int flow_, int cap_, T cost_, bool
            res_)
fc f8             : to(to_), rev(rev_), flow(flow_), cap(cap_),
            res(res_), cost(cost_) {}
83 21     };

ab 00     vector<vector<edge>> g;
f5 16     vector<int> par_idx, par;
20 f1     T inf;
08 a0     vector<T> dist;

ce b2     mcmf(int n) : g(n), par_idx(n), par(n),
            inf(numeric_limits<T>::max()/3) {}

2e 91     void add(int u, int v, int w, T cost) { // de u pra v com cap
            w e custo cost
aa 2f         edge a = edge(v, g[v].size(), 0, w, cost, false);
61 23         edge b = edge(u, g[u].size(), 0, 0, -cost, true);

8b b2         g[u].push_back(a);
c3 c1         g[v].push_back(b);
eb cb     }

9c 8b     vector<T> spfa(int s) { // nao precisa se nao tiver custo
            negativo
e4 87         deque<int> q;
30 3d         vector<bool> is_inside(g.size(), 0);
fa 57         dist = vector<T>(g.size(), inf);

1d a9         dist[s] = 0;
59 a3         q.push_back(s);
56 ec         is_inside[s] = true;

0a 14         while (!q.empty()) {
91 b1             int v = q.front();
ec ce             q.pop_front();
98 48             is_inside[v] = false;

90 76             for (int i = 0; i < g[v].size(); i++) {
4a 9d                 auto [to, rev, flow, cap, res, cost] = g[v][i];
8c e6                 if (flow < cap and dist[v] + cost < dist[to]) {
8a 94                     dist[to] = dist[v] + cost;

```

```

af ed          if (is_inside[to]) continue;
c2 02          if (!q.empty() and dist[to] > dist[q.front()])
    q.push_back(to);
58 b3          else q.push_front(to);
be b5          is_inside[to] = true;
d0 cb          }
19 cb          }
ec cb          }
53 8d          return dist;
9a cb      }
01 2a  bool dijkstra(int s, int t, vector<T>& pot) {
a6 48      priority_queue<pair<T, int>, vector<pair<T, int>>,
    greater<>> q;
07 57      dist = vector<T>(g.size(), inf);
43 a9      dist[s] = 0;
b9 11      q.emplace(0, s);
a0 40      while (q.size()) {
93 91          auto [d, v] = q.top();
38 83          q.pop();
92 68          if (dist[v] < d) continue;
17 76          for (int i = 0; i < g[v].size(); i++) {
4e 9d              auto [to, rev, flow, cap, res, cost] = g[v][i];
be e8              cost += pot[v] - pot[to];
c3 e6              if (flow < cap and dist[v] + cost < dist[to]) {
30 94                  dist[to] = dist[v] + cost;
03 44                  q.emplace(dist[to], to);
e2 88                  par_idx[to] = i, par[to] = v;
7f cb              }
16 cb          }
e4 cb      }
9c 1d      return dist[t] < inf;
5f cb  }

ce 3d  pair<int, T> min_cost_flow(int s, int t, int flow = INF) {
d8 3d      vector<T> pot(g.size(), 0);
56 9e      pot = spfa(s); // mudar algoritmo de caminho minimo aqui

8d d2      int f = 0;
ed ce      T ret = 0;
da 4a      while (f < flow and dijkstra(s, t, pot)) {
06 bd          for (int i = 0; i < g.size(); i++)
f3 d2              if (dist[i] < inf) pot[i] += dist[i];

fd 71          int mn_flow = flow - f, u = t;
8a 04          while (u != s){
97 90              mn_flow = min(mn_flow,

```

```

87 07          g[par[u]][par_idx[u]].cap -
    g[par[u]][par_idx[u]].flow);
ca 3d          u = par[u];
2c cb          }

63 1f          ret += pot[t] * mn_flow;

69 47          u = t;
ac 04          while (u != s) {
bc e0              g[par[u]][par_idx[u]].flow += mn_flow;
3d d9              g[u][g[par[u]][par_idx[u]].rev].flow -= mn_flow;
e3 3d              u = par[u];
74 cb          }

da 04          f += mn_flow;
2b cb          }

7b 15          return make_pair(f, ret);
6d cb      }

    // Opcional: retorna as arestas originais por onde passa flow
    = cap
46 18  vector<pair<int,int>> recover() {
4f 24      vector<pair<int,int>> used;
27 2a      for (int i = 0; i < g.size(); i++) for (edge e : g[i])
fc 58          if(e.flow == e.cap && !e.res) used.push_back({i,
    e.to});
a6 f6          return used;
25 cb      }
69 21 };

```

### 7.33 Prufer code

```

// Traduz de lista de arestas para prufer code
// e vice-versa
// Os vertices tem label de 0 a n-1
// Todo array com n-2 posicoes e valores de
// 0 a n-1 sao prufer codes validos
//
// 0(n)

// d3b324
47 47  vector<int> to_prufer(vector<pair<int, int>> tree) {
70 1f      int n = tree.size()+1;
9f 2c      vector<int> d(n, 0);
e2 4a      vector<vector<int>> g(n);
3e f8      for (auto [a, b] : tree) d[a]++, d[b]++,

```

```

4a f6      g[a].push_back(b), g[b].push_back(a);
a5 c5      vector<int> pai(n, -1);
87 26      queue<int> q; q.push(n-1);
be 40      while (q.size()) {
de be          int u = q.front(); q.pop();
6e 34          for (int v : g[u]) if (v != pai[u])
5a 9c              pai[v] = u, q.push(v);
b1 cb      }
62 39      int idx, x;
f5 89      idx = x = find(d.begin(), d.end(), 1) - d.begin();
31 4b      vector<int> ret;
28 b2      for (int i = 0; i < n-2; i++) {
2e d4          int y = pai[x];
83 e8          ret.push_back(y);
44 66          if (--d[y] == 1 and y < idx) x = y;
52 36          else idx = x = find(d.begin()+idx+1, d.end(), 1) -
d.begin();
2a cb      }
c7 ed      return ret;
d3 cb }

// 765413
8d 4d vector<pair<int, int>> from_prufer(vector<int> p) {
30 45      int n = p.size()+2;
f0 12      vector<int> d(n, 1);
12 65      for (int i : p) d[i]++;
e7 85      p.push_back(n-1);
0e 39      int idx, x;
21 89      idx = x = find(d.begin(), d.end(), 1) - d.begin();
97 1d      vector<pair<int, int>> ret;
05 b0      for (int y : p) {
63 da          ret.push_back({x, y});
16 66          if (--d[y] == 1 and y < idx) x = y;
d7 36          else idx = x = find(d.begin()+idx+1, d.end(), 1) -
d.begin();
0d cb      }
36 ed      return ret;
49 cb }

```

### 7.34 Sack (DSU em arvores)

```

// Responde queries de todas as sub-arvores
// offline
//
// O(n log(n))
// bb361f

```

```

6b 6b int sz[MAX], cor[MAX], cnt[MAX];
4e 04 vector<int> g[MAX];

ef 6d void build(int k, int d=0) {
d0 e8     sz[k] = 1;
e0 01     for (auto& i : g[k]) {
ca 30         build(i, d+1); sz[k] += sz[i];
44 92         if (sz[i] > sz[g[k][0]]) swap(i, g[k][0]);
96 cb     }
88 cb }

f6 74 void compute(int k, int x, bool dont=1) {
41 de     cnt[cor[k]] += x;
d4 82     for (int i = dont; i < g[k].size(); i++)
ac b5         compute(g[k][i], x, 0);
0a cb }

53 dc void solve(int k, bool keep=0) {
ea 32     for (int i = int(g[k].size())-1; i >= 0; i--)
00 b4         solve(g[k][i], !i);
76 4a     compute(k, 1);

// agora cnt[i] tem quantas vezes a cor
// i aparece na sub-arvore do k

5f 83     if (!keep) compute(k, -1, 0);
bb cb }

```

### 7.35 Stable Marriage

```

// Emparelha todos os elementos de A com elementos de B
// de forma que nao exista um par x \in A, y \in B
// e x nao pareado com y tal que x prefira parer com y
// e y prefira parer com x.
//
// a[i] contem os elementos de B ordenados por preferencia de i
// b[j] contem os elementos de A ordenados por preferencia de j
// |A| <= |B|
//
// Retorna um vetor v de tamanho |A| onde v[i] guarda o match de i.
//
// O(|A| * |B|)
// Off8d5

38 38 vector<int> stable_marriage(vector<vector<int>> &a,
vector<vector<int>> &b) {
32 65     int n = a.size(), m = b.size();

```



```

3c 83  assert(a[0].size() == m and b[0].size() == n and n <= m);
19 01  vector<int> match(m, -1), it(n, 0);
65 e6  vector inv_b(m, vector<int>(n));
0d a3  for (int i = 0; i < m; i++) for (int j = 0; j < n; j++)
ae 9f      inv_b[i][b[i][j]] = j;

e4 26  queue<int> q;
70 5a  for (int i = 0; i < n; i++) q.push(i);
e6 40  while (q.size()) {
0c 37      int i = q.front(); q.pop();
ac 4b      int j = a[i][it[i]];

9a 57      if (match[j] == -1) match[j] = i;
48 02      else if (inv_b[j][i] < inv_b[j][match[j]]) {
57 5d          q.emplace(match[j]);
5f e7          it[match[j]]++;
1e f1          match[j] = i;
a9 bc      } else q.emplace(i), it[i]++;
37 cb  }

d7 82  vector<int> ret(n);
69 d7  for (int i = 0; i < m; i++) if (match[i] != -1) ret[match[i]]
    = i;
c6 ed  return ret;
0f cb  }

```

## 7.36 Tarjan para SCC

```

// O(n + m)
// 573bfa

04 04  vector<int> g[MAX];
73 4c  stack<int> s;
d9 a4  int vis[MAX], comp[MAX];
71 3f  int id[MAX];

// se quiser comprimir ciclo ou achar ponte em grafo nao direcionado,
// colocar um if na dfs para nao voltar pro pai da DFS tree
47 f3  int dfs(int i, int& t) {
20 cf      int lo = id[i] = t++;
74 18      s.push(i);
03 0c      vis[i] = 2;

4c 48      for (int j : g[i]) {
63 74          if (!vis[j]) lo = min(lo, dfs(j, t));
f0 99          else if (vis[j] == 2) lo = min(lo, id[j]);
e3 cb      }

```

```

// aresta de i pro pai eh uma ponte (no caso nao direcionado)
41 3d  if (lo == id[i]) while (1) {
77 3c      int u = s.top(); s.pop();
0a 9c      vis[u] = 1, comp[u] = i;
0a 2e      if (u == i) break;
f9 cb  }

7a 25  return lo;
ad cb  }

31 f9  void tarjan(int n) {
3f 6b      int t = 0;
7d 99      for (int i = 0; i < n; i++) vis[i] = 0;

11 3b      for (int i = 0; i < n; i++) if (!vis[i]) dfs(i, t);
57 cb  }

```

## 7.37 Topological Sort

```

// Retorna uma ordenacao topologica de g
// Se g nao for DAG retorna um vetor vazio
//
// O(n + m)
// bdc95e

```

```

04 04  vector<int> g[MAX];

26 b6  vector<int> topo_sort(int n) {
ea 46      vector<int> ret(n, -1), vis(n, 0);

55 f5      int pos = n-1, dag = 1;
a5 36      function<void(int)> dfs = [&](int v) {
79 cc          vis[v] = 1;
77 44          for (auto u : g[v]) {
1c 15              if (vis[u] == 1) dag = 0;
66 53              else if (!vis[u]) dfs(u);
ae cb          }
44 d4          ret[pos--] = v, vis[v] = 2;
2b 21      };

d4 15      for (int i = 0; i < n; i++) if (!vis[i]) dfs(i);

b5 d8      if (!dag) ret.clear();
c2 ed      return ret;
bd cb  }

```

## 7.38 Vertex cover

```
// Encontra o tamanho do vertex cover minimo
// Da pra alterar facil pra achar os vertices
// Parece rodar com < 2 s pra N = 90
//
// O(n * 1.38^n)
// 9c5024

76 76 namespace cover {
ec 5a     const int MAX = 96;
52 04     vector<int> g[MAX];
a4 82     bitset<MAX> bs[MAX];
ca 1a     int n;

01 69     void add(int i, int j) {
75 bd         if (i == j) return;
2a 78         n = max({n, i+1, j+1});
40 20         bs[i][j] = bs[j][i] = 1;
2b cb     }

42 6c     int rec(bitset<MAX> m) {
e5 1a         int ans = 0;
3f 25         for (int x = 0; x < n; x++) if (m[x]) {
ff 00             bitset<MAX> comp;
13 4b             function<void(int)> dfs = [&](int i) {
39 b9                 comp[i] = 1, m[i] = 0;
63 0c                 for (int j : g[i]) if (m[j]) dfs(j);
2a 21             };
63 96             dfs(x);

cb d3                 int ma, deg = -1, cyc = 1;
3a 41                 for (int i = 0; i < n; i++) if (comp[i]) {
99 d0                     int d = (bs[i]&comp).count();
1f 18                     if (d <= 1) cyc = 0;
b9 c1                     if (d > deg) deg = d, ma = i;
fe cb                 }
b5 26                 if (deg <= 2) { // caminho ou ciclo
02 34                     ans += (comp.count() + cyc) / 2;
86 5e                     continue;
49 cb                 }
b2 3f                 comp[ma] = 0;

// ou ta no cover, ou nao ta no cover
a9 1d                 ans += min(1 + rec(comp), deg + rec(comp & ~bs[ma]));
ec cb             }
9e ba     return ans;
}
```

```
45 cb     }
df f5     int solve() {
1a 3c         bitset<MAX> m;
64 60         for (int i = 0; i < n; i++) {
9e 93             m[i] = 1;
1f f9             for (int j = 0; j < n; j++)
19 74                 if (bs[i][j]) g[i].push_back(j);
2d cb         }
7e 4f         return rec(m);
61 cb     }
9c cb }
```

## 7.39 Virtual Tree

```
// Comprime uma arvore dado um conjunto S de vertices, de forma que
// o conjunto de vertices da arvore comprimida contenha S e seja
// minimal e fechado sobre a operacao de LCA
// Se |S| = k, a arvore comprimida tem menos que 2k vertices
// As arestas devirt possuem a distancia do vertice ate o vizinho
// Retorna a raiz da virtual tree
//
// lca::pos deve ser a ordem de visitacao no dfs
// voce pode usar o LCAcomHLD, por exemplo
//
// O(k log(k))
// 42d990

b3 b3 vector<pair<int, int>> virt[MAX];

b3 d4 #warning lembrar de buildar o LCA antes
93 c1 int build_virt(vector<int> v) {
7e b4     auto cmp = [&](int i, int j) { return lca::pos[i] <
lca::pos[j]; };
c5 07     sort(v.begin(), v.end(), cmp);
4b e8     for (int i = v.size()-1; i; i--) v.push_back(lca::lca(v[i],
v[i-1]));
e5 07     sort(v.begin(), v.end(), cmp);
84 d7     v.erase(unique(v.begin(), v.end()), v.end());
d1 37     for (int i = 0; i < v.size(); i++) virt[v[i]].clear();
2d 19     for (int i = 1; i < v.size(); i++) virt[lca::lca(v[i-1],
v[i])].clear();
f6 ad     for (int i = 1; i < v.size(); i++) {
d6 51         int parent = lca::lca(v[i-1], v[i]);
47 29         int d = lca::dist(parent, v[i]);
47 d4 #warning soh to colocando aresta descendo
c7 4d         virt[parent].emplace_back(v[i], d);
80 cb     }
```

```
c1 83     return v[0];
42 cb }
```

## 8 Extra

### 8.1 gethash.sh

```
# Para usar:
# bash gethash.sh arquivo.cpp
echo "" > pref.txt
while IFS= read -r l; do
    echo $l >> pref.txt
    echo $l > line.txt
    hp=$(echo $(bash hash.sh pref.txt 1 1000) | cut -c-2)
    hl=$(echo $(bash hash.sh line.txt 1 1000) | cut -c-2)
    echo -e "$hp $hl $l"
done < "$1"
```

### 8.2 hash.sh

```
# Para usar (hash das linhas [l1, l2]):
# bash hash.sh arquivo.cpp l1 l2
sed -n $2', '$3' p' $1 | sed '/^#w/d' | cpp -dD -P -fpreprocessed | tr
-d '[:space:]' | md5sum | cut -c-6
```

### 8.3 makefile

```
CXX = g++
CXXFLAGS = -fsanitize=address,undefined -fno-omit-frame-pointer -g
-Wall -Wshadow -std=c++17 -Wno-unused-result -Wno-sign-compare
-Wno-char-subscripts #-fuse-ld=gold
```

### 8.4 fastIO.cpp

```
int read_int() {
    bool minus = false;
    int result = 0;
    char ch;
    ch = getchar();
    while (1) {
        if (ch == '-') break;
        if (ch >= '0' && ch <= '9') break;
        ch = getchar();
    }
    if (ch == '-') minus = true;
    else result = ch - '0';
    while (1) {
        ch = getchar();
```

```

        if (ch < '0' || ch > '9') break;
        result = result*10 + (ch - '0');
    }
    if (minus) return -result;
    else return result;
}

```

## 8.5 vimrc

```

set ts=4 si ai sw=4 nu mouse=a undofile
syntax on

```

## 8.6 stress.sh

```

P=a
make ${P} ${P}2 gen || exit 1
for ((i = 1; ; i++)) do
    ./gen $i > in
    ./${P} < in > out
    ./${P}2 < in > out2
    if (! cmp -s out out2) then
        echo "--> entrada:"
        cat in
        echo "--> saida1:"
        cat out
        echo "--> saida2:"
        cat out2
        break;
    fi
    echo $i
done

```

## 8.7 rand.cpp

```

mt19937 rng((int)
    chrono::steady_clock::now().time_since_epoch().count());

int uniform(int l, int r){
    uniform_int_distribution<int> uid(l, r);
    return uid(rng);
}

```

## 8.8 timer.cpp

```

// timer T; T() -> retorna o tempo em ms desde que declarou

```

```

using namespace chrono;
struct timer : high_resolution_clock {
    const time_point start;
    timer(): start(now()) {}
    int operator()() {
        return duration_cast<milliseconds>(now() - start).count();
    }
};

```

## 8.9 debug.cpp

```

void debug_out(string s, int line) { cerr << endl; }
template<typename H, typename... T>
void debug_out(string s, int line, H h, T... t) {
    if (s[0] != ',') cerr << "Line(" << line << ") ";
    do { cerr << s[0]; s = s.substr(1);
    } while (s.size() and s[0] != ',');
    cerr << " = " << h;
    debug_out(s, line, t...);
}
#ifdef DEBUG
#define debug(...) debug_out(__VA_ARGS__, __LINE__, __VA_ARGS__)
#else
#define debug(...) 42
#endif

```

## 8.10 template.cpp

```

#include <bits/stdc++.h>

using namespace std;

#define _ ios_base::sync_with_stdio(0);cin.tie(0);
#define endl '\n'

typedef long long ll;

const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3fll;

int main() { _
    exit(0);
}

```