

# Humuhumunukunukuapua'a

## UFMG

Bruno Monteiro, Emanuel Silva e Bernardo Amorim

<b>Índice</b>	
<b>1 Estruturas</b>	<b>5</b>
1.1 BIT	5
1.2 BIT 2D	5
1.3 BIT com update em range	6
1.4 DSU	7
1.5 Li-Chao Tree	8
1.6 MergeSort Tree	9
1.7 Min queue - deque	10
1.8 Min queue - stack	11
1.9 Order Statistic Set	11
1.10 Range color	12
1.11 RMQ $\langle O(n), O(1) \rangle$ - min queue	12
1.12 SegTreap	13
1.13 SegTree	14
1.14 SegTree 2D Iterativa	15
1.15 SegTree Beats	16
1.16 SegTree Colorida	18
1.17 SegTree Esparsa - Lazy	20
1.18 SegTree Esparsa - $O(q)$ memoria	20
1.19 SegTree Iterativa	21
1.20 SegTree Iterativa com Lazy Propagation	22
1.21 SegTree PA	23
1.22 SegTree Persistente	24
1.23 Sparse Table	25
1.24 Sparse Table Disjunta	25
1.25 Splay Tree	26
1.26 Splay Tree Implicita	27
1.27 Split-Merge Set	29
1.28 Split-Merge Set - Lazy	32
1.29 Sqrt Tree	34
1.30 Treap	35
1.31 Treap Implicita	36

1.32 Treap Persistent Implicita . . . . .	38	2.21 Kruskal . . . . .	56
1.33 Wavelet Tree . . . . .	38	2.22 Kuhn . . . . .	56
<b>2 Grafos</b>	<b>39</b>	2.23 LCA com binary lifting . . . . .	57
2.1 AGM Direcionada . . . . .	39	2.24 LCA com HLD . . . . .	58
2.2 Bellman-Ford . . . . .	40	2.25 LCA com RMQ . . . . .	59
2.3 Block-Cut Tree . . . . .	41	2.26 Line Tree . . . . .	60
2.4 Blossom - matching maximo em grafo geral . . . . .	42	2.27 Link-cut Tree . . . . .	60
2.5 Centro de arvore . . . . .	43	2.28 Link-cut Tree - aresta . . . . .	61
2.6 Centroid . . . . .	43	2.29 Link-cut Tree - vertice . . . . .	63
2.7 Centroid decomposition . . . . .	44	2.30 Max flow com lower bound nas arestas . . . . .	64
2.8 Centroid Tree . . . . .	44	2.31 MinCostMaxFlow . . . . .	65
2.9 Dijkstra . . . . .	45	2.32 Prufer code . . . . .	66
2.10 Dinitz . . . . .	45	2.33 Sack (DSU em arvores) . . . . .	67
2.11 Dominator Tree - Kawakami . . . . .	46	2.34 Tarjan para SCC . . . . .	67
2.12 Euler Path / Euler Cycle . . . . .	47	2.35 Topological Sort . . . . .	68
2.13 Euler Tour Tree . . . . .	48	2.36 Vertex cover . . . . .	68
2.14 Floyd-Warshall . . . . .	51	2.37 Virtual Tree . . . . .	69
2.15 Functional Graph . . . . .	51	<b>3 Problemas</b>	<b>69</b>
2.16 Heavy-Light Decomposition - aresta . . . . .	52	3.1 Algoritmo Hungaro . . . . .	69
2.17 Heavy-Light Decomposition - vertice . . . . .	53	3.2 Algoritmo MO - queries em caminhos de arvore . . . . .	70
2.18 Heavy-Light Decomposition sem Update . . . . .	54	3.3 Angle Range Intersection . . . . .	71
2.19 Isomorfismo de arvores . . . . .	55	3.4 Area da Uniao de Retangulos . . . . .	72
2.20 Kosaraju . . . . .	55	3.5 Area Maxima de Histograma . . . . .	73

3.6	Binomial modular . . . . .	73	3.29	RMQ com Divide and Conquer . . . . .	89
3.7	Closest pair of points . . . . .	74	3.30	Segment Intersection . . . . .	89
3.8	Coloracao de Grafo de Intervalo . . . . .	75	3.31	Sequencia de de Bruijn . . . . .	90
3.9	Conectividade Dinamica . . . . .	75	3.32	Shortest Addition Chain . . . . .	90
3.10	Conectividade Dinamica 2 . . . . .	76	3.33	Simple Polygon . . . . .	90
3.11	Conj. Indep. Maximo com Peso em Grafo de Intervalo . . .	78	3.34	Sweep Direction . . . . .	91
3.12	Distancia maxima entre dois pontos . . . . .	78	3.35	Triangulacao de Delaunay . . . . .	92
3.13	Distinct Range Query . . . . .	79	3.36	Triangulos em Grafos . . . . .	94
3.14	Distinct Range Query com Update . . . . .	79	<b>4</b>	<b>Matematica</b>	<b>94</b>
3.15	Dominator Points . . . . .	80	4.1	2-SAT . . . . .	94
3.16	DP de Dominacao 3D . . . . .	81	4.2	Algoritmo de Euclides estendido . . . . .	95
3.17	Gray Code . . . . .	82	4.3	Avaliacao de Interpolacao . . . . .	95
3.18	Half-plane intersection . . . . .	82	4.4	Berlekamp-Massey . . . . .	96
3.19	Heap Sort . . . . .	83	4.5	Binomial Distribution . . . . .	96
3.20	Inversion Count . . . . .	83	4.6	Convolucao de GCD / LCM . . . . .	97
3.21	LIS - Longest Increasing Subsequence . . . . .	83	4.7	Deteccao de ciclo - Tortoise and Hare . . . . .	97
3.22	LIS2 - Longest Increasing Subsequence . . . . .	84	4.8	Division Trick . . . . .	98
3.23	Mininum Enclosing Circle . . . . .	84	4.9	Eliminacao Gaussiana . . . . .	98
3.24	Minkowski Sum . . . . .	85	4.10	Eliminacao Gaussiana Z2 . . . . .	98
3.25	MO - DSU . . . . .	85	4.11	Equacao Diofantina Linear . . . . .	99
3.26	Mo - numero de distintos em range . . . . .	86	4.12	Exponenciacao rapida . . . . .	100
3.27	Palindromic Factorization . . . . .	87	4.13	Fast Walsh Hadamard Transform . . . . .	100
3.28	Parsing de Expressao . . . . .	88	4.14	FFT . . . . .	100

4.15	Integracao Numerica - Metodo de Simpson 3/8 . . . . .	102	6.4	eertree . . . . .	113
4.16	Inverso Modular . . . . .	102	6.5	KMP . . . . .	114
4.17	Karatsuba . . . . .	102	6.6	Manacher . . . . .	114
4.18	Logaritmo Discreto . . . . .	103	6.7	Min/max suffix/cyclic shift . . . . .	115
4.19	Miller-Rabin . . . . .	103	6.8	String Hashing . . . . .	116
4.20	Pollard's Rho Alg . . . . .	104	6.9	String Hashing - modulo $2^{61} - 1$ . . . . .	116
4.21	Produto de dois long long mod m . . . . .	105	6.10	Suffix Array - $O(n \log n)$ . . . . .	117
4.22	Simplex . . . . .	105	6.11	Suffix Array - $O(n)$ . . . . .	117
4.23	Teorema Chines do Resto . . . . .	106	6.12	Suffix Array Dinamico . . . . .	120
4.24	Totiente . . . . .	106	6.13	Trie . . . . .	122
4.25	Variacoes do crivo de Eratosthenes . . . . .	106	<b>7</b>	<b>Primitivas</b>	<b>123</b>
<b>5</b>	<b>DP</b>	<b>108</b>	7.1	Aritmetica Modular . . . . .	123
5.1	Convex Hull Trick (Rafael) . . . . .	108	7.2	Big Integer . . . . .	124
5.2	Convex Hull Trick Dinamico . . . . .	109	7.3	Matroid . . . . .	128
5.3	Divide and Conquer DP . . . . .	109	7.4	Primitivas de fracao . . . . .	130
5.4	Longest Common Subsequence . . . . .	110	7.5	Primitivas de matriz - exponenciacao . . . . .	131
5.5	Mochila . . . . .	111	7.6	Primitivas Geometricas . . . . .	131
5.6	SOS DP . . . . .	111	7.7	Primitivas Geometricas 3D . . . . .	137
<b>6</b>	<b>Strings</b>	<b>111</b>	7.8	Primitivas Geometricas Inteiras . . . . .	139
6.1	Aho-corasick . . . . .	111	<b>8</b>	<b>Extra</b>	<b>143</b>
6.2	Algoritmo Z . . . . .	112	8.1	fastIO.cpp . . . . .	143
6.3	Automato de Sufixo . . . . .	112	8.2	vimrc . . . . .	143

8.3	timer.cpp	143
8.4	rand.cpp	143
8.5	template.cpp	143
8.6	debug.cpp	143
8.7	stress.sh	144
8.8	makefile	144
8.9	hash.sh	144
8.10	linehash.sh	144

# 1 Estruturas

## 1.1 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

1a8 int n;
7f4 int bit[MAX];
b69 int v[MAX];

0a8 void build() {
b91     bit[0] = 0;
33c     for (int i = 1; i <= n; i++) bit[i] = v[i - 1];
```

```
78a     for (int i = 1; i <= n; i++) {
edf         int j = i + (i & -i);
b8a         if (j <= n) bit[j] += bit[i];
cbb     }

// soma x na posicao p
235 void poe(int x, int p) {
9c7     for (; p <= n; p += p & -p) bit[p] += x;
cbb }

// soma [1, p]
0bf int pref(int p) {
7c9     int ret = 0;
805     for (; p; p -= p & -p) ret += bit[p];
edf     return ret;
cbb }

// soma [a, b]
4ea int query(int a, int b) {
70c     return pref(b) - pref(a - 1);
cbb }

e4a int l_bound(int x) {
1ba     int p = 0;
676     for (int i = MAX2; i+1; i--) if (p + (1<<i) <= n
729         and bit[p + (1<<i)] <= x) x -= bit[p += (1<<i)];
74e     return p;
cbb }
```

## 1.2 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
```

```

a6b template<class T = int> struct bit2d {
acf     vector<T> X;
a84     vector<vector<T>> Y, t;

709     int ub(vector<T>& v, T x) {
dde         return upper_bound(v.begin(), v.end(), x) -
v.begin();
cbb     }
5cb     bit2d(vector<pair<T, T>> v) {
2e1         for (auto [x, y] : v) X.push_back(x);
fd4         sort(X.begin(), X.end());
1ee         X.erase(unique(X.begin(), X.end()), X.end());

d56         t.resize(X.size() + 1);
d12         Y.resize(t.size());
3d0         sort(v.begin(), v.end(), [](auto a, auto b) {
43d             return a.second < b.second; });
961         for (auto [x, y] : v) for (int i = ub(X, x); i <
t.size(); i += i&-i)
b75             if (!Y[i].size() or Y[i].back() != y)
Y[i].push_back(y);

7c7         for (int i = 0; i < t.size(); i++)
t[i].resize(Y[i].size() + 1);
cbb     }

e78     void update(T x, T y, T v) {
2a9         for (int i = ub(X, x); i < t.size(); i += i&-i)
cd2             for (int j = ub(Y[i], y); j < t[i].size(); j
+= j&-j) t[i][j] += v;
cbb     }

5d2     T query(T x, T y) {
966         T ans = 0;
c54         for (int i = ub(X, x); i; i -= i&-i)
4fb             for (int j = ub(Y[i], y); j; j -= j&-j) ans
+= t[i][j];
ba7         return ans;
cbb     }

46d     T query(T x1, T y1, T x2, T y2) {
fcf         return query(x2, y2)-query(x2, y1-1)-query(x1-1,

```

```

y2)+query(x1-1, y1-1);
cbb     }
214 };

```

### 1.3 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

e04 namespace bit {
3ba     ll bit[2][MAX+2];
1a8     int n;

61c     void build(int n2, int* v) {
1e3         n = n2;
535         for (int i = 1; i <= n; i++)
edd             bit[1][min(n+1, i+(i&-i))] += bit[1][i] +=
v[i-1];
cbb     }
637     ll get(int x, int i) {
b73         ll ret = 0;
360         for (; i; i -= i&-i) ret += bit[x][i];
edf         return ret;
cbb     }

20c     void add(int x, int i, ll val) {
503         for (; i <= n; i += i&-i) bit[x][i] += val;
cbb     }

162     ll get2(int p) {
c7c         return get(0, p) * p + get(1, p);
cbb     }

02a     ll query(int l, int r) {
ff5         return get2(r+1) - get2(l);
cbb     }

089     void update(int l, int r, ll x) {
e5f         add(0, l+1, x), add(0, r+2, -x);

```

```
f58      add(1, l+1, -x*1), add(1, r+2, x*(r+1));
cbb      }
214 };
```

## 1.4 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
```

```
8d3 struct dsu {
825     vector<int> id, sz;

b33     dsu(int n) : id(n), sz(n, 1) { iota(id.begin(),
id.end(), 0); }

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

```
440     void unite(int a, int b) {
605         a = find(a), b = find(b);
d54         if (a == b) return;
956         if (sz[a] < sz[b]) swap(a, b);
6d0         sz[a] += sz[b], id[b] = a;
cbb     }
214 };
```

```
// 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
```

```
8d3 struct dsu {
6f7     vector<int> id, sz, bip, c;

5b4     dsu(int n) : id(n), sz(n, 1), bip(n, 1), c(n) {
```

```
db8         iota(id.begin(), id.end(), 0);
cbb     }

ef0     int find(int a) { return a == id[a] ? a :
find(id[a]); }
f30     int color(int a) { return a == id[a] ? c[a] : c[a] ^
color(id[a]); }

440     void unite(int a, int b) {
263         bool change = color(a) == color(b);
605         a = find(a), b = find(b);
a89         if (a == b) {
4ed             if (change) bip[a] = 0;
505             return;
cbb         }

956         if (sz[a] < sz[b]) swap(a, b);
efe         if (change) c[b] = 1;
2cd         sz[a] += sz[b], id[b] = a, bip[a] ^= bip[b];
cbb     }
214 };
```

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

```
8d3 struct dsu {
33c     vector<int> id, sz, ti;

733     dsu(int n) : id(n), sz(n, 1), ti(n, -INF) {
db8         iota(id.begin(), id.end(), 0);
cbb     }

5e6     int find(int a, int t) {
6ba         if (id[a] == a or ti[a] > t) return a;
ea5         return find(id[a], t);
cbb     }
```

```

fa0     void unite(int a, int b, int t) {
84f         a = find(a, t), b = find(b, t);
d54         if (a == b) return;
956         if (sz[a] < sz[b]) swap(a, b);
35d         sz[a] += sz[b], id[b] = a, ti[b] = t;
cbb     }
214 };

// 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

8d3 struct dsu {
825     vector<int> id, sz;
27c     stack<stack<pair<int&, int>>> st;

98d     dsu(int n) : id(n), sz(n, 1) {
1cc         iota(id.begin(), id.end(), 0), st.emplace();
cbb     }

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

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

5cf     void rollback() {
ba9         while(st.top().size()) {
6bf             auto [end, val] = st.top().top();
st.top().pop();
149             end = val;

```

```

cbb     }
25a         st.pop();
cbb     }

ef0     int find(int a) { return a == id[a] ? a :
find(id[a]); }

440     void unite(int a, int b) {
605         a = find(a), b = find(b);
d54         if (a == b) return;
956         if (sz[a] < sz[b]) swap(a, b);
803         save(sz[a]), save(id[b]);
6d0         sz[a] += sz[b], id[b] = a;
cbb     }
214 };

```

## 1.5 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

5b0 template<ll MI = ll(-1e9), ll MA = ll(1e9)> struct
lichao {
b3a     struct line {
12d         ll a, b;
cef         array<int, 2> ch;
fdf         line(ll a_ = 0, ll b_ = LINF) :
423             a(a_), b(b_), ch({-1, -1}) {}
888         ll operator()(ll x) { return a*x + b; }
214     };
17b     vector<line> ln;

df8     int ch(int p, int d) {
e85         if (ln[p].ch[d] == -1) {
9af             ln[p].ch[d] = ln.size();
cdc             ln.emplace_back();

```



```

cbb      }
ef2      return ln[p].ch[d];
cbb      }
021      lichao() { ln.emplace_back(); }

c33      void add(line s, ll l=MI, ll r=MA, int p=0) {
3e3          ll m = (l+r)/2;
911          bool L = s(l) < ln[p](l);
d37          bool M = s(m) < ln[p](m);
03b          bool R = s(r) < ln[p](r);
825          if (M) swap(ln[p], s), swap(ln[p].ch, s.ch);
cac          if (s.b == LINF) return;
f6d          if (L != M) add(s, l, m-1, ch(p, 0));
898          else if (R != M) add(s, m+1, r, ch(p, 1));
cbb      }
092      ll query(int x, ll l=MI, ll r=MA, int p=0) {
11b          ll m = (l+r)/2, ret = ln[p](x);
9db          if (ret == LINF) return ret;
529          if (x < m) return min(ret, query(x, l, m-1,
ch(p, 0)));
81a          return min(ret, query(x, m+1, r, ch(p, 1)));
cbb      }
214 };

```

## 1.6 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, y2) 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

c6c      template <typename T = int> struct ms_tree {
6f7          vector<tuple<T, T, int>> v;
1a8          int n;
5ee          vector<vector<tuple<T, T, int>>> t; // {y, idx, left}
6ae          vector<T> vy;

78c          ms_tree(vector<pair<T, T>>& vv) : n(vv.size()),
t(4*n), vy(n) {
e80              for (int i = 0; i < n; i++)
v.push_back({vv[i].first, vv[i].second, i});
fca              sort(v.begin(), v.end());
224              build(1, 0, n-1);
01a              for (int i = 0; i < n; i++) vy[i] =
get<0>(t[1][i+1]);
cbb          }
dac          ms_tree(vector<T>& vv, bool inv = false) { // inv:
inverte indice e valor
8e8              vector<pair<T, T>> v2;
e1e              for (int i = 0; i < vv.size(); i++)
196                  inv ? v2.push_back({vv[i], i}) :
v2.push_back({i, vv[i]});
cca                  *this = ms_tree(v2);
cbb          }
2c6          void build(int p, int l, int r) {
1d2              t[p].push_back({get<0>(v[l]), get<0>(v[r]), 0});
// {min_x, max_x, 0}
5c8              if (l == r) return t[p].push_back({get<1>(v[l]),
get<2>(v[l]), 0});

```

```

ee4         int m = (l+r)/2;
bd9         build(2*p, l, m), build(2*p+1, m+1, r);

32d         int L = 0, R = 0;
a03         while (t[p].size() <= r-l+1) {
68e             int left = get<2>(t[p].back());
4aa             if (L > m-1 or (R+m+1 <= r and t[2*p+1][1+R]
< t[2*p][1+L])) {
8cf                 t[p].push_back(t[2*p+1][1 + R++]);
da0                 get<2>(t[p].back()) = left;
5e2                 continue;
cbb             }
249             t[p].push_back(t[2*p][1 + L++]);
339             get<2>(t[p].back()) = left+1;
cbb         }
cbb     }

dd3     int get_l(T y) { return lower_bound(vy.begin(),
vy.end(), y) - vy.begin(); }
ebb     int get_r(T y) { return upper_bound(vy.begin(),
vy.end(), y) - vy.begin(); }

f62     int count(T x1, T x2, T y1, T y2) {
902         function<int(int, int, int)> dfs = [&](int p,
int l, int r) {
7c6             if (l == r or x2 < get<0>(t[p][0]) or
get<1>(t[p][0]) < x1) return 0;
2bb             if (x1 <= get<0>(t[p][0]) and
get<1>(t[p][0]) <= x2) return r-l;
784             int nl = get<2>(t[p][1]), nr =
get<2>(t[p][r]);
ebb             return dfs(2*p, nl, nr) + dfs(2*p+1, l-nl,
r-nr);
214         };
7cb         return dfs(1, get_l(y1), get_r(y2));
cbb     }

002     vector<int> report(T x1, T x2, T y1, T y2) {
4b8         vector<int> ret;
85e         function<void(int, int, int)> dfs = [&](int p,
int l, int r) {
882             if (l == r or x2 < get<0>(t[p][0]) or
get<1>(t[p][0]) < x1) return;

```

```

8da             if (x1 <= get<0>(t[p][0]) and
get<1>(t[p][0]) <= x2) {
e00                 for (int i = l; i < r; i++)
ret.push_back(get<1>(t[p][i+1]));
505                 return;
cbb             }
784             int nl = get<2>(t[p][1]), nr =
get<2>(t[p][r]);
194             dfs(2*p, nl, nr), dfs(2*p+1, l-nl, r-nr);
214         };
8ad         dfs(1, get_l(y1), get_r(y2));
edf         return ret;
cbb     }

985     int kth(T y1, T y2, int k) {
902         function<int(int, int, int)> dfs = [&](int p,
int l, int r) {
150             if (k >= r-l) {
941                 k -= r-l;
daa                 return -1;
cbb             }
8da             if (r-l == 1) return get<1>(t[p][l+1]);
784             int nl = get<2>(t[p][1]), nr =
get<2>(t[p][r]);
072             int left = dfs(2*p, nl, nr);
3b6             if (left != -1) return left;
04d             return dfs(2*p+1, l-nl, r-nr);
214         };
7cb         return dfs(1, get_l(y1), get_r(y2));
cbb     }
214 };

```

## 1.7 Min queue - deque

```

// Tudo O(1) amortizado
// c13c57

```

```

1dc template<class T> struct minqueue {
2d8     deque<pair<T, int>> q;

3fc     void push(T x) {
56e         int ct = 1;
953         while (q.size() and x < q.front().first)

```

```

75f         ct += q.front().second, q.pop_front();
987     q.emplace_front(x, ct);
cbb     }
42d     void pop() {
aa2         if (q.back().second > 1) q.back().second--;
c51         else q.pop_back();
cbb     }
ea6     T min() { return q.back().first; }
214 };

```

## 1.8 Min queue - stack

```

// Tudo O(1) amortizado
// fe0cad

```

```

557 template<class T> struct minstack {
81f     stack<pair<T, T>> s;

3fc     void push(T x) {
12b         if (!s.size()) s.push({x, x});
9d9         else s.emplace(x, std::min(s.top().second, x));
cbb     }
4f0     T top() { return s.top().first; }
94a     T pop() {
1f2         T ans = s.top().first;
2eb         s.pop();
ba7         return ans;
cbb     }
614     int size() { return s.size(); }
13b     T min() { return s.top().second; }
214 };

```

```

1dc template<class T> struct minqueue {
cdc     minstack<T> s1, s2;

7cd     void push(T x) { s1.push(x); }
c96     void move() {
d4d         if (s2.size()) return;
d92         while (s1.size()) {
7ae             T x = s1.pop();
489             s2.push(x);
cbb         }

```

```

cbb     }
787     T front() { return move(), s2.top(); }
23a     T pop() { return move(), s2.pop(); }
7f3     int size() { return s1.size()+s2.size(); }
19c     T min() {
cd6         if (!s1.size()) return s2.min();
58e         else if (!s2.size()) return s1.min();
31d         return std::min(s1.min(), s2.min());
cbb     }
214 };

```

## 1.9 Order Statistic Set

```

// Funciona do C++11 pra cima

```

```

774 #include <ext/pb_ds/assoc_container.hpp>
30f #include <ext/pb_ds/tree_policy.hpp>
0d7 using namespace __gnu_pbds;
4fc template <class T>
def     using ord_set = tree<T, null_type, less<T>,
rb_tree_tag,
3a1     tree_order_statistics_node_update>;

// para declarar:
b36 ord_set<int> s;
// coisas do set normal funcionam:
e6f for (auto i : s) cout << i << endl;
738 cout << s.size() << endl;
// k-esimo maior elemento O(log|s|):
// k=0: menor elemento
e46 cout << *s.find_by_order(k) << endl;
// quantos sao menores do que k O(log|s|):
df7 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

```

## 1.10 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
```

```
df6 template<typename T> struct color {  
f0c     set<tuple<int, int, T>> se;  
  
071     vector<tuple<int, int, T>> update(int l, int r, T  
val) {  
9c4         auto it = se.upper_bound({r, INF, val});  
753         if (it != se.begin() and get<1>(*prev(it)) > r) {  
e91             auto [L, R, V] = *--it;  
3f0             se.erase(it);  
bfd             se.emplace(L, r, V), se.emplace(r+1, R, V);  
cbb         }  
d9e         it = se.lower_bound({l, -INF, val});  
516         if (it != se.begin() and get<1>(*prev(it)) >= l)  
{  
e91             auto [L, R, V] = *--it;  
3f0             se.erase(it);  
75a             se.emplace(L, l-1, V), it = se.emplace(l, R,  
V).first;  
cbb         }  
d7b         vector<tuple<int, int, T>> ret;  
7a1         for (; it != se.end() and get<0>(*it) <= r; it =  
se.erase(it))  
8c0             ret.push_back(*it);  
b4a         se.emplace(l, r, val);  
edf         return ret;  
cbb     }  
ff9     T query(int i) {  
c31         auto it = se.upper_bound({i, INF, T()});  
8e7         if (it == se.begin() or get<1>(*--it) < i)  
return -1; // nao tem  
53d         return get<2>(*it);
```

```
cbb     }  
214 };
```

## 1.11 RMQ $\langle O(n), O(1) \rangle$ - min queue

```
// O(n) pra buildar, query O(1)  
// Se tiver varios minimos, retorna  
// o de menor indice  
// bab412
```

```
1a5 template<typename T> struct rmq {  
517     vector<T> v;  
fcc     int n; static const int b = 30;  
70e     vector<int> mask, t;  
  
183     int op(int x, int y) { return v[x] <= v[y] ? x : y; }  
ee1     int msb(int x) { return  
__builtin_clz(1)-__builtin_clz(x); }  
c92     int small(int r, int sz = b) { return  
r-msb(mask[r]&((1<<sz)-1)); }  
6ad     rmq() {}  
43c     rmq(const vector<T>& v_) : v(v_), n(v.size()),  
mask(n), t(n) {  
2e5         for (int i = 0, at = 0; i < n; mask[i++] = at |=  
1) {  
a61             at = (at<<1)&((1<<b)-1);  
c00             while (at and op(i-msb(at&-at), i) == i) at  
^= at&-at;  
cbb         }  
ea4         for (int i = 0; i < n/b; i++) t[i] =  
small(b*i+b-1);  
39d         for (int j = 1; (1<<j) <= n/b; j++) for (int i =  
0; i+(1<<j) <= n/b; i++)  
ba5             t[n/b*j+i] = op(t[n/b*(j-1)+i],  
t[n/b*(j-1)+i+(1<<(j-1))]);  
cbb     }  
e34     int index_query(int l, int r) {  
27b         if (r-l+1 <= b) return small(r, r-l+1);  
e80         int x = l/b+1, y = r/b-1;  
fd3         if (x > y) return op(small(l+b-1), small(r));  
a4e         int j = msb(y-x+1);  
ea3         int ans = op(small(l+b-1), op(t[n/b*j+x],
```

```

        t[n/b*j+y-(1<<j)+1]));
be6         return op(ans, small(r));
cbb     }
093     T query(int l, int r) { return v[index_query(l, r)];
    }
214 };

```

## 1.12 SegTreap

```

// Muda uma posicao do plano, e faz query de operacao
// associativa e comutativa em retangulo
// Mudar ZERO e op
// Esparsa 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

55b const int ZERO = INF;
560 const int op(int l, int r) { return min(l, r); }

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

aa1 template<typename T> struct treap {
3c9     struct node {
b19         node *l, *r;
ee1         int p;
850         pair<ll, ll> idx; // {y, x}
36d         T val, mi;
bc2         node(ll x, ll y, T val_) : l(NULL), r(NULL),
        p(rng()),
1b5         idx(pair(y, x)), val(val_), mi(val) {}
01e         void update() {
d6e             mi = val;
182             if (l) mi = op(mi, l->mi);
b68             if (r) mi = op(mi, r->mi);

```

```

cbb     }
214 };

bb7     node* root;

84b     treap() { root = NULL; }
cec     ~treap() {
609         vector<node*> q = {root};
402         while (q.size()) {
e5d             node* x = q.back(); q.pop_back();
ee9             if (!x) continue;
1c7             q.push_back(x->l), q.push_back(x->r);
bf0             delete x;
cbb         }
cbb     }
225     treap(treap&& t) : treap() { swap(root, t.root); }

bcf     void join(node* l, node* r, node*& i) { // assume
        que l < r
986         if (!l or !r) return void(i = l ? l : r);
80e         if (l->p > r->p) join(l->r, r, l->r), i = l;
fa0         else join(l, r->l, r->l), i = r;
bda         i->update();
cbb     }
c82     void split(node* i, node*& l, node*& r, pair<ll, ll>
        idx) {
26a         if (!i) return void(r = l = NULL);
13c         if (i->idx < idx) split(i->r, i->r, r, idx), l =
        i;
d26         else split(i->l, l, i->l, idx), r = i;
bda         i->update();
cbb     }
d3b     void update(ll x, ll y, T v) {
df9         node *L, *M, *R;
8b2         split(root, M, R, pair(y, x+1)), split(M, L, M,
        pair(y, x));
1e4         if (M) M->val = M->mi = v;
9e5         else M = new node(x, y, v);
69d         join(L, M, M), join(M, R, root);
cbb     }
91b     T query(ll ly, ll ry) {
df9         node *L, *M, *R;

```

```

1c0      split(root, M, R, pair(ry, LINF)), split(M, L,
M, pair(ly, 0));
0f7      T ret = M ? M->mi : ZERO;
69d      join(L, M, M), join(M, R, root);
edf      return ret;
cbb    }
214 };

46a template<typename T> struct segtreap {
c4f      vector<treap<T>> seg;
6e7      vector<int> ch[2];
e4e      ll NX;

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

a71      int get_ch(int i, int d){
e51          if (ch[d][i] == -1) {
2d6              ch[d][i] = seg.size();
23e              seg.emplace_back();
842              ch[0].push_back(-1), ch[1].push_back(-1);
cbb          }
968          return ch[d][i];
cbb      }

10c      T query(ll lx, ll rx, ll ly, ll ry, int p, ll l, ll
r) {
003          if (rx < l or r < lx) return ZERO;
f0f          if (lx <= l and r <= rx) return seg[p].query(ly,
ry);

e6a          ll m = l + (r-l)/2;
354          return op(query(lx, rx, ly, ry, get_ch(p, 0), l,
m),
060              query(lx, rx, ly, ry, get_ch(p, 1), m+1,
r));
cbb      }
f48      T query(ll lx, ll rx, ll ly, ll ry) { return
query(lx, rx, ly, ry, 0, 0, NX); }

249      void update(ll x, ll y, T val, int p, ll l, ll r) {
73c          if (l == r) return seg[p].update(x, y, val);

```

```

e6a          ll m = l + (r-l)/2;
cc5          if (x <= m) update(x, y, val, get_ch(p, 0), l,
m);
5a2          else update(x, y, val, get_ch(p, 1), m+1, r);
980          seg[p].update(x, y, val);
cbb      }
517      void update(ll x, ll y, T val) { update(x, y, val,
0, 0, NX); }
214 };

```

## 1.13 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))

// Oafec1
aa4 namespace seg {
005     ll seg[4*MAX], lazy[4*MAX];
052     int n, *v;

d22     ll build(int p=1, int l=0, int r=n-1) {
3c7         lazy[p] = 0;
6cd         if (l == r) return seg[p] = v[l];
ee4         int m = (l+r)/2;
193         return seg[p] = build(2*p, l, m) + build(2*p+1,
m+1, r);
cbb     }
0d8     void build(int n2, int* v2) {
680         n = n2, v = v2;
6f2         build();
cbb     }
ceb     void prop(int p, int l, int r) {
cdf         seg[p] += lazy[p]*(r-l+1);
2c9         if (l != r) lazy[2*p] += lazy[p], lazy[2*p+1] +=

```

```

        lazy[p];
3c7        lazy[p] = 0;
cbb    }
2c3    ll query(int a, int b, int p=1, int l=0, int r=n-1) {
6b9        prop(p, l, r);
527        if (a <= l and r <= b) return seg[p];
786        if (b < l or r < a) return 0;
ee4        int m = (l+r)/2;
b1f        return query(a, b, 2*p, l, m) + query(a, b,
        2*p+1, m+1, r);
cbb    }
cfb    ll update(int a, int b, int x, int p=1, int l=0, int
        r=n-1) {
6b9        prop(p, l, r);
9a3        if (a <= l and r <= b) {
b94            lazy[p] += x;
6b9            prop(p, l, r);
534            return seg[p];
cbb        }
e9f        if (b < l or r < a) return seg[p];
ee4        int m = (l+r)/2;
fdb        return seg[p] = update(a, b, x, 2*p, l, m) +
7fd            update(a, b, x, 2*p+1, m+1, r);
cbb    }
214 };

// 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
119 int get_left(int a, int b, int val, int p=1, int l=0,
        int r=n-1) {
6b9    prop(p, l, r);
f38    if (b < l or r < a or seg[p] < val) return -1;
205    if (r == l) return l;
ee4    int m = (l+r)/2;
753    int x = get_left(a, b, val, 2*p, l, m);
50e    if (x != -1) return x;
c3c    return get_left(a, b, val, 2*p+1, m+1, r);
cbb }

```

```

// ultima posicao >= val em [a, b] (ou -1 se nao tem)
// 1b71df
992 int get_right(int a, int b, int val, int p=1, int l=0,
        int r=n-1) {
6b9    prop(p, l, r);
f38    if (b < l or r < a or seg[p] < val) return -1;
205    if (r == l) return l;
ee4    int m = (l+r)/2;
1b1    int x = get_right(a, b, val, 2*p+1, m+1, r);
50e    if (x != -1) return x;
6a7    return get_right(a, b, val, 2*p, l, m);
cbb }

// 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
6a9 int lower_bound(int i, ll& val, int p, int l, int r) {
6b9    prop(p, l, r);
6e8    if (r < i) return n;
b5d    if (i <= l and seg[p] < val) {
bff        val -= seg[p];
041        return n;
cbb    }
3ce    if (l == r) return l;
ee4    int m = (l+r)/2;
514    int x = lower_bound(i, val, 2*p, l, m);
ee0    if (x != n) return x;
8b9    return lower_bound(i, val, 2*p+1, m+1, r);
cbb }

```

## 1.14 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

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

0a8 void build() {
919     for (int x = 2*n; x; x--) for (int y = 2*n; y; y--) {
c81         if (x < n) seg[x][y] = seg[2*x][y] +
            seg[2*x+1][y];
fe9         if (y < n) seg[x][y] = seg[x][2*y] +
            seg[x][2*y+1];
cbb     }
cbb }

251 int query(int x1, int y1, int x2, int y2) {
827     int ret = 0, y3 = y1 + n, y4 = y2 + n;
83e     for (x1 += n, x2 += n; x1 <= x2; ++x1 /= 2, --x2 /=
        2)
0f2         for (y1 = y3, y2 = y4; y1 <= y2; ++y1 /= 2, --y2
            /= 2) {
554             if (x1%2 == 1 and y1%2 == 1) ret +=
                seg[x1][y1];
6b0             if (x1%2 == 1 and y2%2 == 0) ret +=
                seg[x1][y2];
c01             if (x2%2 == 0 and y1%2 == 1) ret +=
                seg[x2][y1];
5d4             if (x2%2 == 0 and y2%2 == 0) ret +=
                seg[x2][y2];
cbb         }

edf     return ret;
cbb }

```

```

767 void update(int x, int y, int val) {
66a     int y2 = y += n;
192     for (x += n; x; x /= 2, y = y2) {
970         if (x >= n) seg[x][y] = val;
ba9         else seg[x][y] = seg[2*x][y] + seg[2*x+1][y];

3b1         while (y /= 2) seg[x][y] = seg[x][2*y] +
            seg[x][2*y+1];
cbb     }
cbb }

```

## 1.15 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

```

```

7c6 #define f first
0ab #define s second

f39 namespace beats {
3c9     struct node {
526         int tam;
125         ll sum, lazy; // lazy pra soma
4f3         ll mi1, mi2, mi; // mi = #mi1
c61         ll ma1, ma2, ma; // ma = #ma1

426         node(ll x = 0) {
ba6             sum = mi1 = ma1 = x;
b29             mi2 = LINF, ma2 = -LINF;
62c             mi = ma = tam = 1;
c60             lazy = 0;

```



```

cbb      }
770      node(const node& l, const node& r) {
a95          sum = l.sum + r.sum, tam = l.tam + r.tam;
c60          lazy = 0;
797          if (l.mi1 > r.mi1) {
230              mi1 = r.mi1, mi = r.mi;
ea2              mi2 = min(l.mi1, r.mi2);
dcd          } else if (l.mi1 < r.mi1) {
e34              mi1 = l.mi1, mi = l.mi;
4b3              mi2 = min(r.mi1, l.mi2);
9d9          } else {
a39              mi1 = l.mi1, mi = l.mi+r.mi;
83d              mi2 = min(l.mi2, r.mi2);
cbb          }
cd0          if (l.ma1 < r.ma1) {
6a0              ma1 = r.ma1, ma = r.ma;
96d              ma2 = max(l.ma1, r.ma2);
5f0          } else if (l.ma1 > r.ma1) {
ae0              ma1 = l.ma1, ma = l.ma;
2ca              ma2 = max(r.ma1, l.ma2);
9d9          } else {
db2              ma1 = l.ma1, ma = l.ma+r.ma;
c05              ma2 = max(l.ma2, r.ma2);
cbb          }
cbb      }
4b4      void setmin(ll x) {
55e          if (x >= ma1) return;
463          sum += (x - ma1)*ma;
be5          if (mi1 == ma1) mi1 = x;
0a0          if (mi2 == ma1) mi2 = x;
b81          ma1 = x;
cbb      }
6cb      void setmax(ll x) {
e25          if (x <= mi1) return;
7e8          sum += (x - mi1)*mi;
0bb          if (ma1 == mi1) ma1 = x;
c32          if (ma2 == mi1) ma2 = x;
1ff          mi1 = x;
cbb      }
4cf      void setsum(ll x) {
fe8          mi1 += x, mi2 += x, ma1 += x, ma2 += x;
620          sum += x*tam;

```

```

c46          lazy += x;
cbb      }
214      };

62b      node seg[4*MAX];
052      int n, *v;

93b      node build(int p=1, int l=0, int r=n-1) {
d84          if (l == r) return seg[p] = {v[l]};
ee4          int m = (l+r)/2;
3d6          return seg[p] = {build(2*p, l, m), build(2*p+1,
m+1, r)};
cbb      }
0d8      void build(int n2, int* v2) {
680          n = n2, v = v2;
6f2          build();
cbb      }
ceb      void prop(int p, int l, int r) {
8ce          if (l == r) return;
abd          for (int k = 0; k < 2; k++) {
d07              if (seg[p].lazy)
seg[2*p+k].setsum(seg[p].lazy);
843              seg[2*p+k].setmin(seg[p].ma1);
f79              seg[2*p+k].setmax(seg[p].mi1);
cbb          }
431          seg[p].lazy = 0;
cbb      }
055      pair<pair<ll, ll>, ll> query(int a, int b, int p=1,
int l=0, int r=n-1) {
e07          if (b < l or r < a) return {{LINF, -LINF}, 0};
9be          if (a <= l and r <= b) return {{seg[p].mi1,
seg[p].ma1}, seg[p].sum};
6b9          prop(p, l, r);
ee4          int m = (l+r)/2;
e6f          auto L = query(a, b, 2*p, l, m), R = query(a, b,
2*p+1, m+1, r);
96d          return {{min(L.f.f, R.f.f), max(L.f.s, R.f.s)},
L.s+R.s};
cbb      }
2c8      node updatemin(int a, int b, ll x, int p=1, int l=0,
int r=n-1) {
744          if (b < l or r < a or seg[p].ma1 <= x) return

```

```

seg[p];
309     if (a <= l and r <= b and seg[p].ma2 < x) {
ccd         seg[p].setmin(x);
534         return seg[p];
cbb     }
6b9     prop(p, l, r);
ee4     int m = (l+r)/2;
96a     return seg[p] = {updatemin(a, b, x, 2*p, l, m),
faf         updatemin(a, b, x, 2*p+1, m+1,
r)}};
cbb     }
044     node updatemax(int a, int b, ll x, int p=1, int l=0,
int r=n-1) {
b59         if (b < l or r < a or seg[p].mi1 >= x) return
seg[p];
a9e         if (a <= l and r <= b and seg[p].mi2 > x) {
e8a             seg[p].setmax(x);
534             return seg[p];
cbb         }
6b9         prop(p, l, r);
ee4         int m = (l+r)/2;
ee3         return seg[p] = {updatemax(a, b, x, 2*p, l, m),
bd2             updatemax(a, b, x, 2*p+1, m+1,
r)}};
cbb     }
aee     node updatesum(int a, int b, ll x, int p=1, int l=0,
int r=n-1) {
e9f         if (b < l or r < a) return seg[p];
9a3         if (a <= l and r <= b) {
8f4             seg[p].setsum(x);
534             return seg[p];
cbb         }
6b9         prop(p, l, r);
ee4         int m = (l+r)/2;
7b6         return seg[p] = {updatesum(a, b, x, 2*p, l, m),
ddb             updatesum(a, b, x, 2*p+1, m+1,
r)}};
cbb     }
214 };

```

## 1.16 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

04f struct seg_color {
3c9     struct node {
b19         node *l, *r;
0f9         int cnt;
9ca         ll val, lazy;
277         node() : l(NULL), r(NULL), cnt(0), val(0),
lazy(0) {}
01e         void update() {
d0a             cnt = 0, val = 0;
bc4             for (auto i : {l, r}) if (i) {
c89                 i->prop();
281                 cnt += i->cnt, val += i->val;
cbb             }
cbb         }
a9c         void prop() {
2dd             if (!lazy) return;
3f7             val += lazy*(ll)cnt;
b64             for (auto i : {l, r}) if (i) i->lazy += lazy;
c60             lazy = 0;
cbb         }
214     };

1a8     int n;
9b0     vector<node*> seg;

```

```

6e0     seg_color(vector<pair<int, int>>& v, int c) :
n(v.size()), seg(c, NULL) {
830         for (int i = 0; i < n; i++)
9b7             seg[v[i].second] = insert(seg[v[i].second],
i, v[i].first, 0, n-1);
cbb     }
3c7     ~seg_color() {
dde         queue<node*> q;
3a6         for (auto i : seg) q.push(i);
402         while (q.size()) {
20b             auto i = q.front(); q.pop();
dab             if (!i) continue;
7c7             q.push(i->l), q.push(i->r);
5ce             delete i;
cbb         }
cbb     }

40b     node* insert(node* at, int idx, int val, int l, int
r) {
1a4         if (!at) at = new node();
232         if (l == r) return at->cnt = 1, at->val = val,
at;
ee4         int m = (l+r)/2;
137         if (idx <= m) at->l = insert(at->l, idx, val, l,
m);
3e6         else at->r = insert(at->r, idx, val, m+1, r);
cff         return at->update(), at;
cbb     }

870     ll query(node* at, int a, int b, int l, int r) {
61b         if (!at or b < l or r < a) return 0;
d9f         at->prop();
cb2         if (a <= l and r <= b) return at->val;
ee4         int m = (l+r)/2;
4c4         return query(at->l, a, b, l, m) + query(at->r,
a, b, m+1, r);
cbb     }

e54     ll query(int c, int a, int b) { return query(seg[c],
a, b, 0, n-1); }
91c     void update(node* at, int a, int b, int x, int l,
int r) {
fba         if (!at or b < l or r < a) return;
d9f         at->prop();

```

```

9a3         if (a <= l and r <= b) {
e9a             at->lazy += x;
cb2             return void(at->prop());
cbb         }
ee4         int m = (l+r)/2;
0b0         update(at->l, a, b, x, l, m), update(at->r, a,
b, x, m+1, r);
7b4         at->update();
cbb     }

a40     void update(int c, int a, int b, int x) {
update(seg[c], a, b, x, 0, n-1); }
70c     void paint(node*& from, node*& to, int a, int b, int
l, int r) {
10f         if (to == from or !from or b < l or r < a)
return;
e85         from->prop();
889         if (to) to->prop();
9a3         if (a <= l and r <= b) {
24d             if (!to) {
38f                 to = from;
140                 from = NULL;
505                 return;
cbb             }
ee4             int m = (l+r)/2;
1cb             paint(from->l, to->l, a, b, l, m),
paint(from->r, to->r, a, b, m+1, r);
72d             to->update();
270             delete from;
140             from = NULL;
505             return;
cbb         }
019         if (!to) to = new node();
ee4         int m = (l+r)/2;
1cb             paint(from->l, to->l, a, b, l, m),
paint(from->r, to->r, a, b, m+1, r);
45a             from->update(), to->update();
cbb         }

471     void paint(int c1, int c2, int a, int b) {
paint(seg[c1], seg[c2], a, b, 0, n-1); }
214 };

```

## 1.17 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

aa4 namespace seg {
6de     int seg[MAX], lazy[MAX], R[MAX], L[MAX], ptr;
e9a     int get_l(int i){
3db         if (L[i] == 0) L[i] = ptr++;
a96         return L[i];
cbb     }
943     int get_r(int i){
71b         if (R[i] == 0) R[i] = ptr++;
283         return R[i];
cbb     }

e71     void build() { ptr = 2; }

ceb     void prop(int p, int l, int r) {
b77         if (!lazy[p]) return;
76c         seg[p] = r-l+1 - seg[p];
213         if (l != r) lazy[get_l(p)]^=lazy[p],
        lazy[get_r(p)]^=lazy[p];
3c7         lazy[p] = 0;
cbb     }

158     int query(int a, int b, int p=1, int l=0, int r=N-1)
    {
6b9         prop(p, l, r);
786         if (b < l or r < a) return 0;
527         if (a <= l and r <= b) return seg[p];

ee4         int m = (l+r)/2;
818         return query(a, b, get_l(p), l, m)+query(a, b,
        get_r(p), m+1, r);
```

```
cbb     }

51f     int update(int a, int b, int p=1, int l=0, int
r=N-1) {
6b9         prop(p, l, r);
e9f         if (b < l or r < a) return seg[p];
9a3         if (a <= l and r <= b) {
ab6             lazy[p] ^= 1;
6b9             prop(p, l, r);
534             return seg[p];
cbb         }
ee4         int m = (l+r)/2;
43a         return seg[p] = update(a, b, get_l(p), l,
m)+update(a, b, get_r(p), m+1, r);
cbb     }
214 };
```

## 1.18 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

13d template<typename T> struct seg {
3c9     struct node {
d53         node* ch[2];
970         char d;
ca0         T v;

c4e         T mi;

d4e         node(int d_, T v_, T val) : d(d_), v(v_) {
e71             ch[0] = ch[1] = NULL;
d6e             mi = val;
cbb         }
b32         node(node* x) : d(x->d), v(x->v), mi(x->mi) {
c99             ch[0] = x->ch[0], ch[1] = x->ch[1];
```

```

cbb      }
01e      void update() {
909          mi = numeric_limits<T>::max();
151          for (int i = 0; i < 2; i++) if (ch[i])
b5a              mi = min(mi, ch[i]->mi);
cbb      }
214  };

bb7      node* root;
9c5      char n;

ba7      seg() : root(NULL), n(0) {}
512  ~seg() {
4c0      std::vector<node*> q = {root};
402      while (q.size()) {
e5d          node* x = q.back(); q.pop_back();
ee9          if (!x) continue;
73f          q.push_back(x->ch[0]), q.push_back(x->ch[1]);
bf0          delete x;
cbb      }
cbb  }

1a6      char msb(T v, char l, char r) { // msb in range (l,
r]
8e4          for (char i = r; i > l; i--) if (v>>i&1) return
i;
daa          return -1;
cbb      }

430      void cut(node* at, T v, char i) {
677          char d = msb(v ^ at->v, at->d, i);
23b          if (d == -1) return; // no need to split
ebf          node* nxt = new node(at);
d43          at->ch[v>>d&1] = NULL;
34f          at->ch[!(v>>d&1)] = nxt;
150          at->d = d;
cbb      }

6e5      node* update(node* at, T idx, T val, char i) {
c8c          if (!at) return new node(-1, idx, val);
d67          cut(at, idx, i);
1a2          if (at->d == -1) { // leaf
792              at->mi = val;

```

```

ce6          return at;
cbb      }
b29      bool dir = idx>>at->d&1;
c8f      at->ch[dir] = update(at->ch[dir], idx, val,
at->d-1);
7b4      at->update();
ce6      return at;
cbb      }
85c      void update(T idx, T val) {
8f4          while (idx>>n) n++;
61e          root = update(root, idx, val, n-1);
cbb      }

9d8      T query(node* at, T a, T b, T l, T r, char i) {
df0          if (!at or b < l or r < a) return
numeric_limits<T>::max();
fd3          if (a <= l and r <= b) return at->mi;
841          T m = l + (r-l)/2;
c85          if (at->d < i) {
c59              if ((at->v>>i&1) == 0) return query(at, a,
b, l, m, i-1);
ca4              else return query(at, a, b, m+1, r, i-1);
cbb          }
373          return min(query(at->ch[0], a, b, l, m, i-1),
query(at->ch[1], a, b, m+1, r, i-1));
cbb      }
6f6      T query(T l, T r) { return query(root, l, r, 0,
(1<<n)-1, n-1); }
214  };

```

## 1.19 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

```

```

6a4 int seg[2 * MAX];
1a8 int n;

0a8 void build() {
d15     for (int i = n - 1; i; i--) seg[i] = seg[2*i] +
cbb     seg[2*i+1];
cbb }

4ea int query(int a, int b) {
7c9     int ret = 0;
728     for(a += n, b += n; a <= b; ++a /= 2, --b /= 2) {
4ea         if (a % 2 == 1) ret += seg[a];
244         if (b % 2 == 0) ret += seg[b];
cbb     }
edf     return ret;
cbb }

ff3 void update(int p, int x) {
37d     seg[p += n] = x;
c8c     while (p /= 2) seg[p] = seg[2*p] + seg[2*p+1];
cbb }

```

## 1.20 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

```

```

aa4 namespace seg {
6db     ll seg[2*MAX], lazy[2*MAX];
1a8     int n;

9b3     ll junta(ll a, ll b) {
534         return a+b;

```

```

cbb     }

// soma x na posicao p de tamanho tam
1b4 void poe(int p, ll x, int tam, bool prop=1) {
517     seg[p] += x*tam;
6ae     if (prop and p < n) lazy[p] += x;
cbb }

// atualiza todos os pais da folha p
b1e void sobe(int p) {
d5a     for (int tam = 2; p /= 2; tam *= 2) {
4ca         seg[p] = junta(seg[2*p], seg[2*p+1]);
388         poe(p, lazy[p], tam, 0);
cbb     }
cbb }

// propaga o caminho da raiz ate a folha p
a0a void prop(int p) {
076     int tam = 1 << (LOG-1);
0a8     for (int s = LOG; s; s--, tam /= 2) {
4b1         int i = p >> s;
27c         if (lazy[i]) {
860             poe(2*i, lazy[i], tam);
e38             poe(2*i+1, lazy[i], tam);
b97             lazy[i] = 0;
cbb         }
cbb     }
cbb }

61c void build(int n2, int* v) {
1e3     n = n2;
95f     for (int i = 0; i < n; i++) seg[n+i] = v[i];
c41     for (int i = n-1; i; i--) seg[i] =
junta(seg[2*i], seg[2*i+1]);
f4c     for (int i = 0; i < 2*n; i++) lazy[i] = 0;
cbb }

4f3 ll query(int a, int b) {
b73     ll ret = 0;
b48     for (prop(a+=n), prop(b+=n); a <= b; ++a/=2,
--b/=2) {
a8e         if (a%2 == 1) ret = junta(ret, seg[a]);

```

```

c58         if (b%2 == 0) ret = junta(ret, seg[b]);
cbb     }
edf     return ret;
cbb     }

a28     void update(int a, int b, int x) {
c2d         int a2 = a += n, b2 = b += n, tam = 1;
0ff         for (; a <= b; ++a/=2, --b/=2, tam *= 2) {
32a             if (a%2 == 1) poe(a, x, tam);
9da             if (b%2 == 0) poe(b, x, tam);
cbb         }
0f7         sobe(a2), sobe(b2);
cbb     }
214 };

```

## 1.21 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

dc7 struct seg_pa {
350     struct Data {
8f5         ll sum;
662         ll set_a, set_r, add_a, add_r;
9b7         Data() : sum(0), set_a(LINF), set_r(0),
            add_a(0), add_r(0) {}
214     };
16a     vector<Data> seg;
1a8     int n;

d45     seg_pa(int n_) {
e95         n = n_;
fc3         seg = vector<Data>(4*n);

```

```

cbb     }

ceb     void prop(int p, int l, int r) {
d5a         int tam = r-l+1;
c3f         ll &sum = seg[p].sum, &set_a = seg[p].set_a,
            &set_r = seg[p].set_r,
a1b         &add_a = seg[p].add_a, &add_r = seg[p].add_r;

c02         if (set_a != LINF) {
660             set_a += add_a, set_r += add_r;
06e             sum = set_a*tam + set_r*tam*(tam+1)/2;
579             if (l != r) {
ee4                 int m = (l+r)/2;

886                 seg[2*p].set_a = set_a;
358                 seg[2*p].set_r = set_r;
ed6                 seg[2*p].add_a = seg[2*p].add_r = 0;

f0c                 seg[2*p+1].set_a = set_a + set_r *
                    (m-l+1);
471                 seg[2*p+1].set_r = set_r;
d48                 seg[2*p+1].add_a = seg[2*p+1].add_r = 0;
cbb             }
823             set_a = LINF, set_r = 0;
953             add_a = add_r = 0;
105         } else if (add_a or add_r) {
18b             sum += add_a*tam + add_r*tam*(tam+1)/2;
579             if (l != r) {
ee4                 int m = (l+r)/2;

ff0                 seg[2*p].add_a += add_a;
ec0                 seg[2*p].add_r += add_r;

06c                 seg[2*p+1].add_a += add_a + add_r *
                    (m-l+1);
a6d                 seg[2*p+1].add_r += add_r;
cbb             }
953             add_a = add_r = 0;
cbb         }
cbb     }

0b7     int inter(pair<int, int> a, pair<int, int> b) {

```

```

98c         if (a.first > b.first) swap(a, b);
eef         return max(0, min(a.second, b.second) - b.first
+ 1);
cbb     }
be1     ll set(int a, int b, ll aa, ll rr, int p, int l, int
r) {
6b9         prop(p, l, r);
457         if (b < l or r < a) return seg[p].sum;
9a3         if (a <= l and r <= b) {
91c             seg[p].set_a = aa;
774             seg[p].set_r = rr;
6b9             prop(p, l, r);
254             return seg[p].sum;
cbb         }
ee4         int m = (l+r)/2;
963         int tam_l = inter({l, m}, {a, b});
c34         return seg[p].sum = set(a, b, aa, rr, 2*p, l, m)
+
365             set(a, b, aa + rr * tam_l, rr, 2*p+1, m+1,
r);
cbb     }
f55     void update_set(int l, int r, ll aa, ll rr) {
6f7         set(l, r, aa, rr, 1, 0, n-1);
cbb     }
5f6     ll add(int a, int b, ll aa, ll rr, int p, int l, int
r) {
6b9         prop(p, l, r);
457         if (b < l or r < a) return seg[p].sum;
9a3         if (a <= l and r <= b) {
359             seg[p].add_a += aa;
1ee             seg[p].add_r += rr;
6b9             prop(p, l, r);
254             return seg[p].sum;
cbb         }
ee4         int m = (l+r)/2;
963         int tam_l = inter({l, m}, {a, b});
586         return seg[p].sum = add(a, b, aa, rr, 2*p, l, m)
+
695             add(a, b, aa + rr * tam_l, rr, 2*p+1, m+1,
r);
cbb     }
848     void update_add(int l, int r, ll aa, ll rr) {

```

```

afa         add(l, r, aa, rr, 1, 0, n-1);
cbb     }
f45     ll query(int a, int b, int p, int l, int r) {
6b9         prop(p, l, r);
786         if (b < l or r < a) return 0;
e9a         if (a <= l and r <= b) return seg[p].sum;
ee4         int m = (l+r)/2;
b1f         return query(a, b, 2*p, l, m) + query(a, b,
2*p+1, m+1, r);
cbb     }
bfc     ll query(int l, int r) { return query(l, r, 1, 0,
n-1); }
214 };

```

## 1.22 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

```

```

54a const int MAX = 1e5+10, UPD = 1e5+10, LOG = 18;
6de const int MAXS = 2*MAX+UPD*LOG;

```

```

f6e namespace perseg {
bd6     ll seg[MAXS];
f4e     int rt[UPD], L[MAXS], R[MAXS], cnt, t;
052     int n, *v;

3c4     ll build(int p, int l, int r) {
6cd         if (l == r) return seg[p] = v[l];
855         L[p] = cnt++, R[p] = cnt++;
ee4         int m = (l+r)/2;
275         return seg[p] = build(L[p], l, m) + build(R[p],
m+1, r);

```



```

cbb    }
0d8    void build(int n2, int* v2) {
680        n = n2, v = v2;
856        rt[0] = cnt++;
c50        build(0, 0, n-1);
cbb    }
f45    ll query(int a, int b, int p, int l, int r) {
786        if (b < l or r < a) return 0;
527        if (a <= l and r <= b) return seg[p];
ee4        int m = (l+r)/2;
1ed        return query(a, b, L[p], l, m) + query(a, b,
R[p], m+1, r);
cbb    }
182    ll query(int a, int b, int tt) {
c13        return query(a, b, rt[tt], 0, n-1);
cbb    }
bb3    ll update(int a, int x, int lp, int p, int l, int r)
{
747        if (l == r) return seg[p] = seg[lp]+x;
ee4        int m = (l+r)/2;
ab8        if (a <= m)
b48            return seg[p] = update(a, x, L[lp],
L[p]=cnt++, l, m) + seg[R[p]=R[lp]];
8a9        return seg[p] = seg[L[p]=L[lp]] + update(a, x,
R[lp], R[p]=cnt++, m+1, r);
cbb    }
6f6    int update(int a, int x, int tt=t) {
ab3        update(a, x, rt[tt], rt[++t]=cnt++, 0, n-1);
e0d        return t;
cbb    }
214 };

```

## 1.23 Sparse Table

```

// Resolve RMQ
// MAX2 = log(MAX)
//
// Complexidades:
// build - O(n log(n))
// query - O(1)
// 7aa4c9

```

```

cca namespace sparse {
710     int m[MAX2][MAX], n;
61c     void build(int n2, int* v) {
1e3         n = n2;
78e         for (int i = 0; i < n; i++) m[0][i] = v[i];
a1c         for (int j = 1; (1<<j) <= n; j++) for (int i =
0; i+(1<<j) <= n; i++)
5d5             m[j][i] = min(m[j-1][i],
m[j-1][i+(1<<(j-1))]);
cbb     }
4ea     int query(int a, int b) {
ee5         int j = __builtin_clz(1) - __builtin_clz(b-a+1);
dc3         return min(m[j][a], m[j][b-(1<<j)+1]);
cbb     }
cbb }

```

## 1.24 Sparse Table Disjunta

```

// Resolve qualquer operacao associativa
// MAX2 = log(MAX)
//
// Complexidades:
// build - O(n log(n))
// query - O(1)
// fd81ae

```

```

cca namespace sparse {
9bf     int m[MAX2][2*MAX], n, v[2*MAX];
5f7     int op(int a, int b) { return min(a, b); }
0d8     void build(int n2, int* v2) {
1e3         n = n2;
df4         for (int i = 0; i < n; i++) v[i] = v2[i];
a84         while (n&(n-1)) n++;
3d2         for (int j = 0; (1<<j) < n; j++) {
1c0             int len = 1<<j;
d9b             for (int c = len; c < n; c += 2*len) {
332                 m[j][c] = v[c], m[j][c-1] = v[c-1];
668                 for (int i = c+1; i < c+len; i++)
m[j][i] = op(m[j][i-1], v[i]);
432                 for (int i = c-2; i >= c-len; i--)
m[j][i] = op(v[i], m[j][i+1]);
cbb             }

```

```

cbb      }
cbb      }
9e3      int query(int l, int r) {
f13          if (l == r) return v[l];
e6d          int j = __builtin_clz(1) - __builtin_clz(l^r);
d67          return op(m[j][l], m[j][r]);
cbb      }
cbb }

```

## 1.25 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

```

```

538 template<typename T> struct splaytree {
3c9     struct node {
183         node *ch[2], *p;
e4d         int sz;
f48         T val;
da0         node(T v) {
696             ch[0] = ch[1] = p = NULL;
a26             sz = 1;
250             val = v;
cbb         }
01e         void update() {
a26             sz = 1;
c7c             for (int i = 0; i < 2; i++) if (ch[i]) {
d5f                 sz += ch[i]->sz;
cbb             }
cbb         }
214     };

bb7     node* root;

fbc     splaytree() { root = NULL; }
214     splaytree(const splaytree& t) {
cbf         throw logic_error("Nao copiar a splaytree!");
cbb     }

```

```

891 ~splaytree() {
609     vector<node*> q = {root};
402     while (q.size()) {
e5d         node* x = q.back(); q.pop_back();
ee9         if (!x) continue;
73f         q.push_back(x->ch[0]), q.push_back(x->ch[1]);
bf0         delete x;
cbb     }
cbb }

94f void rotate(node* x) { // x vai ficar em cima
d9b     node *p = x->p, *pp = p->p;
ecf     if (pp) pp->ch[pp->ch[1] == p] = x;
286     bool d = p->ch[0] == x;
d63     p->ch[!d] = x->ch[d], x->ch[d] = p;
bad     if (p->ch[!d]) p->ch[!d]->p = p;
fc2     x->p = pp, p->p = x;
1ea     p->update(), x->update();
cbb }

3fa node* splay(node* x) {
a39     if (!x) return x;
4ea     root = x;
3cf     while (x->p) {
d9b         node *p = x->p, *pp = p->p;
359         if (!pp) return rotate(x), x; // zig
e3c         if ((pp->ch[0] == p)^(p->ch[0] == x))
a2b             rotate(x), rotate(x); // zigzag
4b2         else rotate(p), rotate(x); // zigzig
cbb     }
ea5     return x;
cbb }

313 node* insert(T v, bool lb=0) {
b64     if (!root) return lb ? NULL : root = new node(v);
002     node *x = root, *last = NULL;;
31e     while (1) {
5d7         bool d = x->val < v;
0fd         if (!d) last = x;
c2e         if (x->val == v) break;
c16         if (x->ch[d]) x = x->ch[d];
4e6         else {
dea             if (lb) break;
055             x->ch[d] = new node(v);

```

```

99c         x->ch[d]->p = x;
30e         x = x->ch[d];
c2b         break;
cbb     }
cbb     }
0b6     splay(x);
61c     return lb ? splay(last) : x;
cbb     }
c0c     int size() { return root ? root->sz : 0; }
2ca     int count(T v) { return insert(v, 1) and root->val
== v; }

111     node* lower_bound(T v) { return insert(v, 1); }
26b     void erase(T v) {
446         if (!count(v)) return;
bce         node *x = root, *l = x->ch[0];
268         if (!l) {
8b1             root = x->ch[1];
32e             if (root) root->p = NULL;
8f3             return delete x;
cbb         }
5e7         root = l, l->p = NULL;
902         while (l->ch[1]) l = l->ch[1];
bab         splay(l);
f0e         l->ch[1] = x->ch[1];
7d9         if (l->ch[1]) l->ch[1]->p = l;
bf0         delete x;
62a         l->update();
cbb     }
24a     int order_of_key(T v) {
62b         if (!lower_bound(v)) return root ? root->sz : 0;
1cc         return root->ch[0] ? root->ch[0]->sz : 0;
cbb     }
db6     node* find_by_order(int k) {
084         if (k >= size()) return NULL;
52f         node* x = root;
31e         while (1) {
20f             if (x->ch[0] and x->ch[0]->sz >= k+1) x =
x->ch[0];
4e6             else {
a1c                 if (x->ch[0]) k -= x->ch[0]->sz;
1dc                 if (!k) return splay(x);
eb8                 k--, x = x->ch[1];

```

```

cbb         }
cbb     }
cbb     }
19c     T min() {
52f         node* x = root;
6f6         while (x->ch[0]) x = x->ch[0]; // max -> ch[1]
3e9         return splay(x)->val;
cbb     }
214 };

```

## 1.26 Splay Tree Implicita

```

// vector da NASA
// Um pouco mais rapido q a treap
// 0 construtor a partir do vector
// eh linear, todas as outras operacoes
// custom 0(log(n)) amortizado
// a3575a

081 template<typename T> struct splay {
3c9     struct node {
183         node *ch[2], *p;
e4d         int sz;
875         T val, sub, lazy;
aa6         bool rev;
da0         node(T v) {
696             ch[0] = ch[1] = p = NULL;
a26             sz = 1;
1e4             sub = val = v;
c60             lazy = 0;
b67             rev = false;
cbb         }
a9c         void prop() {
0ec             if (lazy) {
924                 val += lazy, sub += lazy*sz;
091                 if (ch[0]) ch[0]->lazy += lazy;
1a8                 if (ch[1]) ch[1]->lazy += lazy;
cbb             }
1bb             if (rev) {
80a                 swap(ch[0], ch[1]);
628                 if (ch[0]) ch[0]->rev ^= 1;
adc                 if (ch[1]) ch[1]->rev ^= 1;

```

```

cbb        }
a32        lazy = 0, rev = 0;
cbb        }
01e        void update() {
0c3            sz = 1, sub = val;
c7c            for (int i = 0; i < 2; i++) if (ch[i]) {
05f                ch[i]->prop();
d5f                sz += ch[i]->sz;
4a1                sub += ch[i]->sub;
cbb            }
cbb        }
214    };

bb7    node* root;

5d9    splay() { root = NULL; }
9b1    splay(node* x) {
4ea        root = x;
32e        if (root) root->p = NULL;
cbb    }
1b7    splay(vector<T> v) { // O(n)
950        root = NULL;
806        for (T i : v) {
2a0            node* x = new node(i);
bd1            x->ch[0] = root;
37a            if (root) root->p = x;
4ea            root = x;
a0a            root->update();
cbb        }
cbb    }
a9e    splay(const splay& t) {
e62        throw logic_error("Nao copiar a splay!");
cbb    }
5ab    ~splay() {
609        vector<node*> q = {root};
402        while (q.size()) {
e5d            node* x = q.back(); q.pop_back();
ee9            if (!x) continue;
73f            q.push_back(x->ch[0]), q.push_back(x->ch[1]);
bf0            delete x;
cbb        }
cbb    }

```

```

73c    int size(node* x) { return x ? x->sz : 0; }
94f    void rotate(node* x) { // x vai ficar em cima
d9b        node *p = x->p, *pp = p->p;
ecf        if (pp) pp->ch[pp->ch[1] == p] = x;
286        bool d = p->ch[0] == x;
d63        p->ch[!d] = x->ch[d], x->ch[d] = p;
bad        if (p->ch[!d]) p->ch[!d]->p = p;
fc2        x->p = pp, p->p = x;
1ea        p->update(), x->update();
cbb    }
6a0    node* splaya(node* x) {
a39        if (!x) return x;
be6        root = x, x->update();
3cf        while (x->p) {
d9b            node *p = x->p, *pp = p->p;
359            if (!pp) return rotate(x), x; // zig
e3c            if ((pp->ch[0] == p)^(p->ch[0] == x))
a2b                rotate(x), rotate(x); // zigzag
4b2            else rotate(p), rotate(x); // zigzig
cbb        }
ea5        return x;
cbb    }
a7f    node* find(int v) {
a2e        if (!root) return NULL;
52f        node *x = root;
6cd        int key = 0;
31e        while (1) {
857            x->prop();
ba1            bool d = key + size(x->ch[0]) < v;
877            if (key + size(x->ch[0]) != v and x->ch[d]) {
15e                if (d) key += size(x->ch[0])+1;
30e                x = x->ch[d];
9af            } else break;
cbb        }
152        return splaya(x);
cbb    }
c0c    int size() { return root ? root->sz : 0; }
c26    void join(splay<T>& l) { // assume que l < *this
690        if (!size()) swap(root, l.root);
579        if (!size() or !l.size()) return;
bee        node* x = l.root;

```

```

31e         while (l) {
857             x->prop();
34d             if (!x->ch[1]) break;
bd8             x = x->ch[1];
cbb         }
147         l.splaya(x), root->prop(), root->update();
42b         x->ch[1] = root, x->ch[1]->p = x;
0aa         root = l.root, l.root = NULL;
a0a         root->update();
cbb     }
5ed     node* split(int v) { // retorna os elementos < v
398         if (v <= 0) return NULL;
060         if (v >= size()) {
f87             node* ret = root;
950             root = NULL;
8c9             ret->update();
edf             return ret;
cbb         }
adc         find(v);
a59         node* l = root->ch[0];
4df         root->ch[0] = NULL;
5a3         if (l) l->p = NULL;
a0a         root->update();
792         return l;
cbb     }
511     T& operator [](int i) {
9d4         find(i);
ae0         return root->val;
cbb     }
231     void push_back(T v) { // O(1)
a01         node* r = new node(v);
0de         r->ch[0] = root;
b11         if (root) root->p = r;
b13         root = r, root->update();
cbb     }
b7a     T query(int l, int r) {
95f         splay<T> M(split(r+1));
5ff         splay<T> L(M.split(l));
d1c         T ans = M.root->sub;
49c         M.join(L), join(M);
ba7         return ans;
cbb     }

```

```

41f     void update(int l, int r, T s) {
95f         splay<T> M(split(r+1));
5ff         splay<T> L(M.split(l));
996         M.root->lazy += s;
49c         M.join(L), join(M);
cbb     }
8c1     void reverse(int l, int r) {
95f         splay<T> M(split(r+1));
5ff         splay<T> L(M.split(l));
945         M.root->rev ^= 1;
49c         M.join(L), join(M);
cbb     }
2fb     void erase(int l, int r) {
95f         splay<T> M(split(r+1));
5ff         splay<T> L(M.split(l));
dcc         join(L);
cbb     }
214 };

```

## 1.27 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

```

```

2dc     template<typename T, bool MULTI=false, typename
SIZE_T=int> struct sms {
3c9         struct node {
b19             node *l, *r;
15f             SIZE_T cnt;
658             node() : l(NULL), r(NULL), cnt(0) {}
01e             void update() {
a01                 cnt = 0;
d8a                 if (l) cnt += l->cnt;
e49                 if (r) cnt += r->cnt;
cbb             }
214     };

```

```

bb7      node* root;
fd0      T N;

f34      sms() : root(NULL), N(0) {}
83b      sms(T v) : sms() { while (v >= N) N = 2*N+1; }
5e1      sms(const sms& t) : root(NULL), N(t.N) {
3af          for (SIZE_T i = 0; i < t.size(); i++) {
a0f              T at = t[i];
e6d              SIZE_T qt = t.count(at);
a43              insert(at, qt);
f42              i += qt-1;
cbb          }
cbb      }
a96      sms(initializer_list<T> v) : sms() { for (T i : v)
insert(i); }
2dd      ~sms() {
609          vector<node*> q = {root};
402          while (q.size()) {
e5d              node* x = q.back(); q.pop_back();
ee9              if (!x) continue;
1c7              q.push_back(x->l), q.push_back(x->r);
bf0              delete x;
cbb          }
cbb      }

fdc      friend void swap(sms& a, sms& b) {
49e          swap(a.root, b.root), swap(a.N, b.N);
cbb      }
83e      sms& operator =(const sms& v) {
768          sms tmp = v;
420          swap(tmp, *this);
357          return *this;
cbb      }
d06      SIZE_T size() const { return root ? root->cnt : 0; }
17f      SIZE_T count(node* x) const { return x ? x->cnt : 0; }
}

75a      void clear() {
0a0          sms tmp;
4ac          swap(*this, tmp);
cbb      }
a06      void expand(T v) {
bc3          for (; N < v; N = 2*N+1) if (root) {

```

```

63c          node* nroot = new node();
956          nroot->l = root;
897          root = nroot;
a0a          root->update();
cbb      }
cbb      }

b14      node* insert(node* at, T idx, SIZE_T qt, T l, T r) {
1a4          if (!at) at = new node();
893          if (l == r) {
435              at->cnt += qt;
beb              if (!MULTI) at->cnt = 1;
ce6              return at;
cbb          }
841          T m = l + (r-l)/2;
a02          if (idx <= m) at->l = insert(at->l, idx, qt, l,
m);
8d9          else at->r = insert(at->r, idx, qt, m+1, r);
cff          return at->update(), at;
cbb      }
cf7      void insert(T v, SIZE_T qt=1) { // insere 'qt'
ocurrencias de 'v'
882          if (qt <= 0) return erase(v, -qt);
72b          assert(v >= 0);
f52          expand(v);
5e9          root = insert(root, v, qt, 0, N);
cbb      }

f06      node* erase(node* at, T idx, SIZE_T qt, T l, T r) {
28c          if (!at) return at;
54b          if (l == r) at->cnt = at->cnt < qt ? 0 : at->cnt
- qt;
4e6          else {
841              T m = l + (r-l)/2;
281              if (idx <= m) at->l = erase(at->l, idx, qt,
l, m);
ba1              else at->r = erase(at->r, idx, qt, m+1, r);
7b4              at->update();
cbb          }
135          if (!at->cnt) delete at, at = NULL;
ce6          return at;
cbb      }

```

```

43d void erase(T v, SIZE_T qt=1) { // remove 'qt'
    ocorrencias de 'v'
9c3 if (v < 0 or v > N or !qt) return;
9dc if (qt < 0) insert(v, -qt);
b1d root = erase(root, v, qt, 0, N);
cbb }
8d6 void erase_all(T v) { // remove todos os 'v'
347 if (v < 0 or v > N) return;
9f2 root = erase(root, v,
    numeric_limits<SIZE_T>::max(), 0, N);
cbb }

0fe SIZE_T count(node* at, T a, T b, T l, T r) const {
61b if (!at or b < l or r < a) return 0;
0fe if (a <= l and r <= b) return at->cnt;
841 T m = l + (r-l)/2;
84a return count(at->l, a, b, l, m) + count(at->r,
    a, b, m+1, r);
cbb }
0a9 SIZE_T count(T v) const { return count(root, v, v,
    0, N); }
ffc SIZE_T order_of_key(T v) { return count(root, 0,
    v-1, 0, N); }
df2 SIZE_T lower_bound(T v) { return order_of_key(v); }

e68 const T operator [] (SIZE_T i) const { // i-esimo
    menor elemento
809 assert(i >= 0 and i < size());
c43 node* at = root;
4a5 T l = 0, r = N;
40c while (l < r) {
841 T m = l + (r-l)/2;
5c2 if (count(at->l) > i) at = at->l, r = m;
4e6 else {
b4a i -= count(at->l);
ded at = at->r; l = m+1;
cbb }
cbb }
792 return l;
cbb }

78c node* merge(node* l, node* r) {

```

```

347 if (!l or !r) return l ? l : r;
504 if (!l->l and !l->r) { // folha
599 if (MULTI) l->cnt += r->cnt;
55d delete r;
792 return l;
cbb }
f58 l->l = merge(l->l, r->l), l->r = merge(l->r,
    r->r);
f4f l->update(), delete r;
792 return l;
cbb }
f59 void merge(sms& s) { // mergeia dois sets
068 if (N > s.N) swap(*this, s);
785 expand(s.N);
938 root = merge(root, s.root);
ee2 s.root = NULL;
cbb }

dc6 node* split(node*& x, SIZE_T k) {
7ca if (k <= 0 or !x) return NULL;
6d0 node* ret = new node();
386 if (!x->l and !x->r) x->cnt -= k, ret->cnt += k;
4e6 else {
85e if (k <= count(x->l)) ret->l = split(x->l,
    k);
4e6 else {
06f ret->r = split(x->r, k - count(x->l));
cfd swap(x->l, ret->l);
cbb }
674 ret->update(), x->update();
cbb }
d5b if (!x->cnt) delete x, x = NULL;
edf return ret;
cbb }
02b void split(SIZE_T k, sms& s) { // pega os 'k' menores
e63 s.clear();
6e5 s.root = split(root, min(k, size()));
e3c s.N = N;
cbb }
// pega os menores que 'k'
131 void split_val(T k, sms& s) { split(order_of_key(k),
    s); }

```

```
214 };
```

## 1.28 Split-Merge Set - Lazy

```
// Representa um conjunto de inteiros nao negativos
// Todas as operacoes custam O(log(N)),
// em que N = maior elemento do set,
// exceto o merge e o insert_range, que custa O(log(N))
// amortizado
// Usa O(min(N, n log(N))) de memoria, sendo 'n' o
// numero de elementos distintos no set
// 3828d0

fb1 template<typename T> struct sms {
3c9     struct node {
b19         node *l, *r;
0f9         int cnt;
393         bool flip;
0fa         node() : l(NULL), r(NULL), cnt(0), flip(0) {}
01e         void update() {
a01             cnt = 0;
d8a             if (l) cnt += l->cnt;
e49             if (r) cnt += r->cnt;
cbb         }
214     };

aee     void prop(node* x, int size) {
bb3         if (!x or !x->flip) return;
f2c         x->flip = 0;
fec         x->cnt = size - x->cnt;
23f         if (size > 1) {
641             if (!x->l) x->l = new node();
756             if (!x->r) x->r = new node();
ddd             x->l->flip ^= 1;
0ff             x->r->flip ^= 1;
cbb         }
cbb     }

bb7     node* root;
fd0     T N;

f34     sms() : root(NULL), N(0) {}
```

```
83b     sms(T v) : sms() { while (v >= N) N = 2*N+1; }
bdd     sms(sms& t) : root(NULL), N(t.N) {
dc5         for (int i = 0; i < t.size(); i++) insert(t[i]);
cbb     }
a96     sms(initializer_list<T> v) : sms() { for (T i : v)
insert(i); }

b2a     void destroy(node* r) {
685         vector<node*> q = {r};
402         while (q.size()) {
e5d             node* x = q.back(); q.pop_back();
ee9             if (!x) continue;
1c7             q.push_back(x->l), q.push_back(x->r);
bf0             delete x;
cbb         }
cbb     }
b58     ~sms() { destroy(root); }

fdc     friend void swap(sms& a, sms& b) {
49e         swap(a.root, b.root), swap(a.N, b.N);
cbb     }
83e     sms& operator =(const sms& v) {
768         sms tmp = v;
420         swap(tmp, *this);
357         return *this;
cbb     }
ff8     int count(node* x, T size) {
a66         if (!x) return 0;
793         prop(x, size);
ead         return x->cnt;
cbb     }
4fe     int size() { return count(root, N+1); }
75a     void clear() {
0a0         sms tmp;
4ac         swap(*this, tmp);
cbb     }
a06     void expand(T v) {
bc3         for (; N < v; N = 2*N+1) if (root) {
edf             prop(root, N+1);
63c             node* nroot = new node();
956             nroot->l = root;
897             root = nroot;
a0a             root->update();
```



```

cbb      }
cbb      }

fde      node* insert(node* at, T idx, T l, T r) {
1a4          if (!at) at = new node();
5ae          else prop(at, r-l+1);
893          if (l == r) {
44b              at->cnt = 1;
ce6              return at;
cbb          }
841          T m = l + (r-l)/2;
95a          if (idx <= m) at->l = insert(at->l, idx, l, m);
018          else at->r = insert(at->r, idx, m+1, r);
cff          return at->update(), at;
cbb      }
c27      void insert(T v) {
72b          assert(v >= 0);
f52          expand(v);
7f2          root = insert(root, v, 0, N);
cbb      }

393      node* erase(node* at, T idx, T l, T r) {
28c          if (!at) return at;
553          prop(at, r-l+1);
4be          if (l == r) at->cnt = 0;
4e6          else {
841              T m = l + (r-l)/2;
d2d              if (idx <= m) at->l = erase(at->l, idx, l,
m);
f3c              else at->r = erase(at->r, idx, m+1, r);
7b4              at->update();
cbb          }
ce6          return at;
cbb      }
26b      void erase(T v) {
347          if (v < 0 or v > N) return;
980          root = erase(root, v, 0, N);
cbb      }

b4f      int count(node* at, T a, T b, T l, T r) {
61b          if (!at or b < l or r < a) return 0;
553          prop(at, r-l+1);

```

```

0fe          if (a <= l and r <= b) return at->cnt;
841          T m = l + (r-l)/2;
84a          return count(at->l, a, b, l, m) + count(at->r,
a, b, m+1, r);
cbb      }
b36      int count(T v) { return count(root, v, v, 0, N); }
eb0      int order_of_key(T v) { return count(root, 0, v-1,
0, N); }
fb8      int lower_bound(T v) { return order_of_key(v); }

dec      const T operator [](int i) { // i-esimo menor
elemento
809          assert(i >= 0 and i < size());
c43          node* at = root;
4a5          T l = 0, r = N;
40c          while (l < r) {
553              prop(at, r-l+1);
841              T m = l + (r-l)/2;
4e7              if (count(at->l, m-l+1) > i) at = at->l, r =
m;
4e6              else {
e6c                  i -= count(at->l, r-m);
ded                  at = at->r; l = m+1;
cbb              }
cbb          }
792          return l;
cbb      }

63d      node* merge(node* a, node* b, T tam) {
c48          if (!a or !b) return a ? a : b;
10e          prop(a, tam), prop(b, tam);
abd          if (b->cnt == tam) swap(a, b);
bb3          if (tam == 1 or a->cnt == tam) {
a9e              destroy(b);
3f5              return a;
cbb          }
c14          a->l = merge(a->l, b->l, tam>>1), a->r =
merge(a->r, b->r, tam>>1);
496          a->update(), delete b;
3f5          return a;
cbb      }
f59      void merge(sms& s) { // mergeia dois sets

```

```

068         if (N > s.N) swap(*this, s);
785         expand(s.N);
707         root = merge(root, s.root, N+1);
ee2         s.root = NULL;
cbb     }

f76     node* split(node*& x, int k, T tam) {
7ca         if (k <= 0 or !x) return NULL;
e3b         prop(x, tam);
6d0         node* ret = new node();
37b         if (tam == 1) x->cnt = 0, ret->cnt = 1;
4e6         else {
b20             if (k <= count(x->l, tam>>1)) ret->l =
split(x->l, k, tam>>1);
4e6             else {
5d8                 ret->r = split(x->r, k - count(x->l,
tam>>1), tam>>1);
cfd                 swap(x->l, ret->l);
cbb             }
674             ret->update(), x->update();
cbb         }
edf         return ret;
cbb     }

049     void split(int k, sms& s) { // pega os 'k' menores
e63         s.clear();
eb6         s.root = split(root, min(k, size()), N+1);
e3c         s.N = N;
cbb     }

// pega os menores que 'k'
131     void split_val(T k, sms& s) { split(order_of_key(k),
s); }

ecf     void flip(node*& at, T a, T b, T l, T r) {
1a4         if (!at) at = new node();
5ae         else prop(at, r-l+1);
9a3         if (a <= l and r <= b) {
747             at->flip ^= 1;
553             prop(at, r-l+1);
505             return;
cbb         }
cc9         if (r < a or b < l) return;
841         T m = l + (r-l)/2;

```

```

2a1         flip(at->l, a, b, l, m), flip(at->r, a, b, m+1,
r);
7b4         at->update();
cbb     }
1ee     void flip(T l, T r) { // flipa os valores em [l, r]
63e         assert(l >= 0 and l <= r);
34b         expand(r);
de7         flip(root, l, r, 0, N);
cbb     }

// complemento considerando que o universo eh [0,
lim]

042     void complement(T lim) {
2e9         assert(lim >= 0);
95c         if (lim > N) expand(lim);
11a         flip(root, 0, lim, 0, N);
0a0         sms tmp;
180         split_val(lim+1, tmp);
4ac         swap(*this, tmp);
cbb     }

0eb     void insert_range(T l, T r) { // insere todo os
valores em [l, r]
0a0         sms tmp;
5fa         tmp.flip(l, r);
7f7         merge(tmp);
cbb     }
214 };

```

## 1.29 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

97a namespace sqrtTree {
052     int n, *v;
ec7     int pref[4][MAX], sul[4][MAX], getl[4][MAX],
entre[4][MAX], sz[4];

```

```

5f7     int op(int a, int b) { return min(a, b); }
c72     inline int getblk(int p, int i) { return
(i-getl[p][i])/sz[p]; }
2c6     void build(int p, int l, int r) {
bc8         if (l+1 >= r) return;
368         for (int i = l; i <= r; i++) getl[p][i] = l;
f16         for (int L = l; L <= r; L += sz[p]) {
191             int R = min(L+sz[p]-1, r);
89c             pref[p][L] = v[L], sulf[p][R] = v[R];
59f             for (int i = L+1; i <= R; i++) pref[p][i] =
op(pref[p][i-1], v[i]);
d9a             for (int i = R-1; i >= L; i--) sulf[p][i] =
op(v[i], sulf[p][i+1]);
221             build(p+1, L, R);
cbb         }
695         for (int i = 0; i <= sz[p]; i++) {
ca5             int at = entre[p][l+i*sz[p]+i] =
sulf[p][l+i*sz[p]];
759             for (int j = i+1; j <= sz[p]; j++)
entre[p][l+i*sz[p]+j] = at =
23a                 op(at, sulf[p][l+j*sz[p]]);
cbb         }
cbb     }
0d8     void build(int n2, int* v2) {
680         n = n2, v = v2;
44c         for (int p = 0; p < 4; p++) sz[p] = n2 =
sqrt(n2);
c50         build(0, 0, n-1);
cbb     }
9e3     int query(int l, int r) {
792         if (l+1 >= r) return l == r ? v[l] : op(v[l],
v[r]);
1ba         int p = 0;
4ba         while (getblk(p, l) == getblk(p, r)) p++;
9e4         int ans = sulf[p][l], a = getblk(p, l)+1, b =
getblk(p, r)-1;
8bf         if (a <= b) ans = op(ans,
entre[p][getl[p][l]+a*sz[p]+b]);
dea         return op(ans, pref[p][r]);
cbb     }
cbb }

```

## 1.30 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

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

aa1 template<typename T> struct treap {
3c9     struct node {
b19         node *l, *r;
284         int p, sz;
36d         T val, mi;
4c7         node(T v) : l(NULL), r(NULL), p(rng()), sz(1),
val(v), mi(v) {}
01e         void update() {
a26             sz = 1;
d6e             mi = val;
bd7             if (l) sz += l->sz, mi = min(mi, l->mi);
a54             if (r) sz += r->sz, mi = min(mi, r->mi);
cbb         }
214     };

bb7     node* root;

84b     treap() { root = NULL; }
2d8     treap(const treap& t) {
465         throw logic_error("Nao copiar a treap!");
cbb     }
cec     ~treap() {
609         vector<node*> q = {root};
402         while (q.size()) {
e5d             node* x = q.back(); q.pop_back();
ee9             if (!x) continue;
1c7             q.push_back(x->l), q.push_back(x->r);
bf0             delete x;
cbb         }

```

```

cbb      }

73c      int size(node* x) { return x ? x->sz : 0; }
b2b      int size() { return size(root); }
bcf      void join(node* l, node* r, node&& i) { // assume
que l < r
986          if (!l or !r) return void(i = l ? l : r);
80e          if (l->p > r->p) join(l->r, r, l->r), i = l;
fa0          else join(l, r->l, r->l), i = r;
bda          i->update();
cbb      }
ece      void split(node* i, node&& l, node&& r, T v) {
26a          if (!i) return void(r = l = NULL);
f05          if (i->val < v) split(i->r, i->r, r, v), l = i;
807          else split(i->l, l, i->l, v), r = i;
bda          i->update();
cbb      }
3fc      void split_leq(node* i, node&& l, node&& r, T v) {
26a          if (!i) return void(r = l = NULL);
181          if (i->val <= v) split_leq(i->r, i->r, r, v), l
= i;
58f          else split_leq(i->l, l, i->l, v), r = i;
bda          i->update();
cbb      }
e13      int count(node* i, T v) {
6b4          if (!i) return 0;
352          if (i->val == v) return 1;
8d0          if (v < i->val) return count(i->l, v);
4d0          return count(i->r, v);
cbb      }
26d      void index_split(node* i, node&& l, node&& r, int v,
int key = 0) {
26a          if (!i) return void(r = l = NULL);
c10          if (key + size(i->l) < v) index_split(i->r,
i->r, r, v, key+size(i->l)+1), l = i;
e5a          else index_split(i->l, l, i->l, v, key), r = i;
bda          i->update();
cbb      }
a1f      int count(T v) {
e06          return count(root, v);
cbb      }
c27      void insert(T v) {

```

```

980          if (count(v)) return;
031          node *L, *R;
d42          split(root, L, R, v);
585          node* at = new node(v);
59f          join(L, at, L);
a28          join(L, R, root);
cbb      }
26b      void erase(T v) {
df9          node *L, *M, *R;
b6b          split_leq(root, M, R, v), split(M, L, M, v);
f17          if (M) delete M;
f38          M = NULL;
a28          join(L, R, root);
cbb      }
e77      void meld(treap& t) { // segmented merge
4a6          node *L = root, *R = t.root;
950          root = NULL;
6b1          while (L or R) {
fe2              if (!L or (L and R and L->mi > R->mi))
std::swap(L, R);
5e1              if (!R) join(root, L, root), L = NULL;
3c9              else if (L->mi == R->mi) {
a76                  node* LL;
439                  split(L, LL, L, R->mi+1);
359                  delete LL;
9d9              } else {
a76                  node* LL;
537                  split(L, LL, L, R->mi);
dbb                  join(root, LL, root);
cbb              }
cbb          }
689          t.root = NULL;
cbb      }
214 };

```

## 1.31 Treap Implicita

```

// Todas as operacoes custam
// O(log(n)) com alta probabilidade
// 63ba4d

```

```

878 mt19937 rng((int)

```

```

        chrono::steady_clock::now().time_since_epoch().count());

aa1 template<typename T> struct treap {
3c9     struct node {
b19         node *l, *r;
284         int p, sz;
875         T val, sub, lazy;
aa6         bool rev;
8dc         node(T v) : l(NULL), r(NULL), p(rng()), sz(1),
        val(v), sub(v), lazy(0), rev(0) {}
a9c         void prop() {
0ec             if (lazy) {
924                 val += lazy, sub += lazy*sz;
b87                 if (l) l->lazy += lazy;
d3b                 if (r) r->lazy += lazy;
cbb             }
1bb             if (rev) {
e4f                 swap(l, r);
dc8                 if (l) l->rev ^= 1;
f2f                 if (r) r->rev ^= 1;
cbb             }
a32             lazy = 0, rev = 0;
cbb         }
01e         void update() {
0c3             sz = 1, sub = val;
a09             if (l) l->prop(), sz += l->sz, sub += l->sub;
095             if (r) r->prop(), sz += r->sz, sub += r->sub;
cbb         }
214     };

bb7     node* root;

84b     treap() { root = NULL; }
2d8     treap(const treap& t) {
465         throw logic_error("Nao copiar a treap!");
cbb     }
cec     ~treap() {
609         vector<node*> q = {root};
402         while (q.size()) {
e5d             node* x = q.back(); q.pop_back();
ee9             if (!x) continue;
1c7             q.push_back(x->l), q.push_back(x->r);

```

```

bf0         delete x;
cbb     }
cbb }

73c     int size(node* x) { return x ? x->sz : 0; }
b2b     int size() { return size(root); }
bcf     void join(node* l, node* r, node*& i) { // assume
que l < r
986         if (!l or !r) return void(i = l ? l : r);
161         l->prop(), r->prop();
80e         if (l->p > r->p) join(l->r, r, l->r), i = l;
fa0         else join(l, r->l, r->l), i = r;
bda         i->update();
cbb     }
a20     void split(node* i, node*& l, node*& r, int v, int
key = 0) {
26a         if (!i) return void(r = l = NULL);
c89         i->prop();
5bd         if (key + size(i->l) < v) split(i->r, i->r, r,
v, key+size(i->l)+1), l = i;
219         else split(i->l, l, i->l, v, key), r = i;
bda         i->update();
cbb     }
231     void push_back(T v) {
2e0         node* i = new node(v);
7ab         join(root, i, root);
cbb     }
b7a     T query(int l, int r) {
df9         node *L, *M, *R;
dca         split(root, M, R, r+1), split(M, L, M, l);
d43         T ans = M->sub;
69d         join(L, M, M), join(M, R, root);
ba7         return ans;
cbb     }
41f     void update(int l, int r, T s) {
df9         node *L, *M, *R;
dca         split(root, M, R, r+1), split(M, L, M, l);
8f6         M->lazy += s;
69d         join(L, M, M), join(M, R, root);
cbb     }
8c1     void reverse(int l, int r) {
df9         node *L, *M, *R;

```

```

dca      split(root, M, R, r+1), split(M, L, M, 1);
66a      M->rev ^= 1;
69d      join(L, M, M), join(M, R, root);
cbb    }
214 };

```

## 1.32 Treap Persistent Implicita

```

// Todas as operacoes custam
// O(log(n)) com alta probabilidade
// fb8013

6cf mt19937_64 rng((int)
    chrono::steady_clock::now().time_since_epoch().count());

3c9 struct node {
b19     node *l, *r;
f14     ll sz, val, sub;
304     node(ll v) : l(NULL), r(NULL), sz(1), val(v), sub(v)
    {}
c12     node(node* x) : l(x->l), r(x->r), sz(x->sz),
    val(x->val), sub(x->sub) {}
01e     void update() {
0c3         sz = 1, sub = val;
77e         if (l) sz += l->sz, sub += l->sub;
d6e         if (r) sz += r->sz, sub += r->sub;
124         sub %= MOD;
cbb     }
214 };

bc9 ll size(node* x) { return x ? x->sz : 0; }
761 void update(node* x) { if (x) x->update(); }
828 node* copy(node* x) { return x ? new node(x) : NULL; }

b02 node* join(node* l, node* r) {
e1f     if (!l or !r) return l ? copy(l) : copy(r);
48b     node* ret;
49f     if (rng() % (size(l) + size(r)) < size(l)) {
7eb         ret = copy(l);
cc1         ret->r = join(ret->r, r);
9d9     } else {
4c5         ret = copy(r);

```

```

551         ret->l = join(l, ret->l);
cbb     }
74f     return update(ret), ret;
cbb }

723 void split(node* x, node*& l, node*& r, ll v, ll key =
    0) {
421     if (!x) return void(l = r = NULL);
b4b     if (key + size(x->l) < v) {
72f         l = copy(x);
d70         split(l->r, l->r, r, v, key+size(l->l)+1);
9d9     } else {
303         r = copy(x);
417         split(r->l, l, r->l, v, key);
cbb     }
da2     update(l), update(r);
cbb }

f9e vector<node*> treap;

139 void init(const vector<ll>& v) {
bbd     treap = {NULL};
969     for (auto i : v) treap[0] = join(treap[0], new
    node(i));
cbb }

```

## 1.33 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 - 0(log(sigma))
// kth - 0(log(sigma))
// sum - 0(log(sigma))
// sumk - 0(log(sigma))
// 782344

597 int n, v[MAX];
578 vector<int> esq[4*(MAXN-MINN)], pref[4*(MAXN-MINN)];

f8d void build(int b = 0, int e = n, int p = 1, int l =
MINN, int r = MAXN) {
58f     int m = (l+r)/2; esq[p].push_back(0);
     pref[p].push_back(0);
f2f     for (int i = b; i < e; i++) {
6b9         esq[p].push_back(esq[p].back()+(v[i]<=m));
26f         pref[p].push_back(pref[p].back()+v[i]);
cbb     }
8ce     if (l == r) return;
3a7     int m2 = stable_partition(v+b, v+e, [=](int
i){return i <= m;}) - v;
347     build(b, m2, 2*p, l, m), build(m2, e, 2*p+1, m+1, r);
cbb }

540 int count(int i, int j, int x, int y, int p = 1, int l =
MINN, int r = MAXN) {
2ad     if (y < l or r < x) return 0;
4db     if (x <= l and r <= y) return j-i;
ddc     int m = (l+r)/2, ei = esq[p][i], ej = esq[p][j];
0a5     return count(ei, ej, x, y, 2*p, l, m)+count(i-ei,
j-ej, x, y, 2*p+1, m+1, r);
cbb }

f62 int kth(int i, int j, int k, int p=1, int l = MINN, int
r = MAXN) {
3ce     if (l == r) return l;
ddc     int m = (l+r)/2, ei = esq[p][i], ej = esq[p][j];
585     if (k <= ej-ei) return kth(ei, ej, k, 2*p, l, m);
28b     return kth(i-ei, j-ej, k-(ej-ei), 2*p+1, m+1, r);
cbb }

f2c int sum(int i, int j, int x, int y, int p = 1, int l =
MINN, int r = MAXN) {

```

```

2ad     if (y < l or r < x) return 0;
2a9     if (x <= l and r <= y) return pref[p][j]-pref[p][i];
ddc     int m = (l+r)/2, ei = esq[p][i], ej = esq[p][j];
43b     return sum(ei, ej, x, y, 2*p, l, m) + sum(i-ei,
j-ej, x, y, 2*p+1, m+1, r);
cbb }

b84 int sumk(int i, int j, int k, int p = 1, int l = MINN,
int r = MAXN) {
8a1     if (l == r) return l*k;
ddc     int m = (l+r)/2, ei = esq[p][i], ej = esq[p][j];
50c     if (k <= ej-ei) return sumk(ei, ej, k, 2*p, l, m);
4c9     return pref[2*p][ej]-pref[2*p][ei]+sumk(i-ei, j-ej,
k-(ej-ei), 2*p+1, m+1, r);
cbb }

```

## 2 Grafos

### 2.1 AGM Direcionada

```

// Fala o menor custo para selecionar arestas tal que
// o vertice 'r' alcance todos
// Se nao tem como, retorna LINF
//
// 0(m log(n))
// dc345b

```

```

3c9 struct node {
f31     pair<ll, int> val;
4e4     ll lazy;
b19     node *l, *r;
f93     node() {}
c53     node(pair<int, int> v) : val(v), lazy(0), l(NULL),
r(NULL) {}

a9c     void prop() {
768         val.first += lazy;
b87         if (l) l->lazy += lazy;
d3b         if (r) r->lazy += lazy;
c60         lazy = 0;

```

```

cbb    }
214 };
de5 void merge(node*& a, node* b) {
c11     if (!a) swap(a, b);
802     if (!b) return;
626     a->prop(), b->prop();
d04     if (a->val > b->val) swap(a, b);
4b0     merge(rand()%2 ? a->l : a->r, b);
cbb }
d01 pair<ll, int> pop(node*& R) {
e8f     R->prop();
22e     auto ret = R->val;
af0     node* tmp = R;
3f3     merge(R->l, R->r);
6c9     R = R->l;
3e4     if (R) R->lazy -= ret.first;
7c3     delete tmp;
edf     return ret;
cbb }
6f6 void apaga(node* R) { if (R) apaga(R->l), apaga(R->r),
    delete R; }

f13 ll dmst(int n, int r, vector<pair<pair<int, int>, int>>&
ar) {
94e     vector<int> p(n); iota(p.begin(), p.end(), 0);
a23     function<int(int)> find = [&](int k) { return
p[k]==k?k:p[k]=find(p[k]); };
2d7     vector<node*> h(n);
56f     for (auto e : ar) merge(h[e.first.second], new
node({e.second, e.first.first}));
fd1     vector<int> pai(n, -1), path(n);
66e     pai[r] = r;
04b     ll ans = 0;

603     for (int i = 0; i < n; i++) { // vai conectando todo
        mundo
2a3         int u = i, at = 0;
cae         while (pai[u] == -1) {
daa             if (!h[u]) { // nao tem
947                 for (auto i : h) apaga(i);
77c                 return LINF;
cbb             }

```

```

167         path[at++] = u, pai[u] = i;
55e         auto [mi, v] = pop(h[u]);
64c         ans += mi;

5e2         if (pai[u = find(v)] == i) { // ciclo
86f             while (find(v = path[--at]) != u)
621                 merge(h[u], h[v]), h[v] = NULL,
p[find(v)] = u;
57a             pai[u] = -1;
cbb         }
cbb     }
cbb }
947 for (auto i : h) apaga(i);
ba7 return ans;
cbb }

```

## 2.2 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

14e int n, m;
248 int d[MAX];
e93 vector<pair<int, int>> ar; // vetor de arestas
9e2 vector<int> w;           // peso das arestas

6be bool bellman_ford(int a) {
8ec     for (int i = 0; i < n; i++) d[i] = INF;
8a8     d[a] = 0;

4e3     for (int i = 0; i <= n; i++)
891         for (int j = 0; j < m; j++) {
6e4             if (d[ar[j].second] > d[ar[j].first] + w[j])
{
705                 if (i == n) return 1;

```



```

e93          d[ar[j].second] = d[ar[j].first] + w[j];
cbb      }
cbb  }

bb3      return 0;
cbb  }

```

## 2.3 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

d10 struct block_cut_tree {
d8e     vector<vector<int>> g, blocks, tree;
43b     vector<vector<pair<int, int>>> edgblocks;
4ce     stack<int> s;
6c0     stack<pair<int, int>> s2;
2bb     vector<int> id, art, pos;

763     block_cut_tree(vector<vector<int>> g_) : g(g_) {
af1         int n = g.size();

```

```

37a         id.resize(n, -1), art.resize(n), pos.resize(n);
6f2         build();
cbb     }

df6     int dfs(int i, int& t, int p = -1) {
cf0         int lo = id[i] = t++;
18e         s.push(i);

827         if (p != -1) s2.emplace(i, p);
53f         for (int j : g[i]) if (j != p and id[j] != -1)
s2.emplace(i, j);

cac         for (int j : g[i]) if (j != p) {
9a3             if (id[j] == -1) {
121                 int val = dfs(j, t, i);
0c3                 lo = min(lo, val);

588                 if (val >= id[i]) {
66a                     art[i]++;
483                     blocks.emplace_back(1, i);
110                     while (blocks.back().back() != j)
138                         blocks.back().push_back(s.top()), s.pop();

128                         edgblocks.emplace_back(1, s2.top()),
s2.pop();
47e                         while (edgblocks.back().back() !=
pair(j, i))
bce                             edgblocks.back().push_back(s2.top()), s2.pop();
cbb                             }
// if (val > id[i]) aresta i-j eh ponte

cbb     }
328         else lo = min(lo, id[j]);
cbb     }

3bd         if (p == -1 and art[i]) art[i]--;
253         return lo;
cbb     }

0a8     void build() {
6bb         int t = 0;

```

```

abf         for (int i = 0; i < g.size(); i++) if (id[i] ==
-1) dfs(i, t, -1);

56c         tree.resize(blocks.size());
f7d         for (int i = 0; i < g.size(); i++) if (art[i])
965             pos[i] = tree.size(), tree.emplace_back();

973         for (int i = 0; i < blocks.size(); i++) for (int
j : blocks[i]) {
403             if (!art[j]) pos[j] = i;
101             else tree[i].push_back(pos[j]),
tree[pos[j]].push_back(i);
cbb         }
cbb     }
214 };

```

## 2.4 Blossom - matching maximo em grafo geral

```

// O(n^3)
// Se for bipartido, nao precisa da funcao
// 'contract', e roda em O(nm)
// 4426a4

042 vector<int> g[MAX];
128 int match[MAX]; // match[i] = com quem i esta matchzado
ou -1
1f1 int n, pai[MAX], base[MAX], vis[MAX];
26a queue<int> q;

107 void contract(int u, int v, bool first = 1) {
165     static vector<bool> blossom;
fbe     static int l;
418     if (first) {
a47         blossom = vector<bool>(n, 0);
042         vector<bool> teve(n, 0);
ddf         int k = u; l = v;
31e         while (1) {
297             teve[k = base[k]] = 1;
116             if (match[k] == -1) break;
dfa             k = pai[match[k]];
cbb         }
d31         while (!teve[l = base[l]]) l = pai[match[l]];

```

```

cbb     }
2e9     while (base[u] != 1) {
e29         blossom[base[u]] = blossom[base[match[u]]] = 1;
8fa         pai[u] = v;
0b0         v = match[u];
a51         u = pai[match[u]];
cbb     }
71c     if (!first) return;
95e     contract(v, u, 0);
6ee     for (int i = 0; i < n; i++) if (bloss[base[i]]) {
594         base[i] = 1;
ca7         if (!vis[i]) q.push(i);
29a         vis[i] = 1;
cbb     }
cbb }

f10 int getpath(int s) {
88f     for (int i = 0; i < n; i++) base[i] = i, pai[i] =
-1, vis[i] = 0;
ded     vis[s] = 1; q = queue<int>(); q.push(s);
402     while (q.size()) {
be1         int u = q.front(); q.pop();
bdc         for (int i : g[u]) {
7a2             if (base[i] == base[u] or match[u] == i)
continue;
e35             if (i == s or (match[i] != -1 and
pai[match[i]] != -1))
4f2                 contract(u, i);
e2e             else if (pai[i] == -1) {
545                 pai[i] = u;
f6a                 if (match[i] == -1) return i;
818                 i = match[i];
29d                 vis[i] = 1; q.push(i);
cbb             }
cbb         }
cbb     }
daa     return -1;
cbb }

83f int blossom() {
1a4     int ans = 0;
315     memset(match, -1, sizeof(match));

```

```

2e3     for (int i = 0; i < n; i++) if (match[i] == -1)
f76         for (int j : g[i]) if (match[j] == -1) {
1bc             match[i] = j;
f1d             match[j] = i;
0df             ans++;
c2b             break;
cbb         }
da8     for (int i = 0; i < n; i++) if (match[i] == -1) {
7e3         int j = getpath(i);
5f2         if (j == -1) continue;
0df         ans++;
3a0         while (j != -1) {
ef0             int p = pai[j], pp = match[p];
348             match[p] = j;
fe9             match[j] = p;
55d             j = pp;
cbb         }
cbb     }
ba7     return ans;
cbb }

```

## 2.5 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
//
// 0(n)
// cladeb

042 vector<int> g[MAX];
df1 int d[MAX], par[MAX];

544 pair<int, vector<int>> center() {
a95     int f, df;
36d     function<void(int)> dfs = [&] (int v) {
d47         if (d[v] > df) f = v, df = d[v];
e68         for (int u : g[v]) if (u != par[v])
1a5             d[u] = d[v] + 1, par[u] = v, dfs(u);
214     };

1b0     f = df = par[0] = -1, d[0] = 0;

```

```

41e     dfs(0);
c2d     int root = f;
0f6     f = df = par[root] = -1, d[root] = 0;
14e     dfs(root);

761     vector<int> c;
87e     while (f != -1) {
999         if (d[f] == df/2 or d[f] == (df+1)/2)
            c.push_back(f);
19c         f = par[f];
cbb     }

00f     return {df, c};
cbb }

```

## 2.6 Centroid

```

// Computa os 2 centroids da arvore
//
// 0(n)
// e16075

97a int n, subsize[MAX];
042 vector<int> g[MAX];

98f void dfs(int k, int p=-1) {
bd2     subsize[k] = 1;
6e5     for (int i : g[k]) if (i != p) {
801         dfs(i, k);
2e3         subsize[k] += subsize[i];
cbb     }
cbb }

2e8 int centroid(int k, int p=-1, int size=-1) {
e73     if (size == -1) size = subsize[k];
8df     for (int i : g[k]) if (i != p) if (subsize[i] >
        size/2)
bab         return centroid(i, k, size);
839     return k;
cbb }

f20 pair<int, int> centroids(int k=0) {

```

```

051     dfs(k);
909     int i = centroid(k), i2 = i;
8dd     for (int j : g[i]) if (2*subsize[j] == subsize[k])
        i2 = j;
0cb     return {i, i2};
cbb }

```

## 2.7 Centroid decomposition

```

// decomp(0, k) computa numero de caminhos com 'k' arestas
// Mudar depois do comentario
//
// O(n log(n))
// fe2541

042 vector<int> g[MAX];
ba8 int sz[MAX], rem[MAX];

747 void dfs(vector<int>& path, int i, int l=-1, int d=0) {
547     path.push_back(d);
75f     for (int j : g[i]) if (j != l and !rem[j]) dfs(path,
        j, i, d+1);
cbb }

071 int dfs_sz(int i, int l=-1) {
02c     sz[i] = 1;
e5c     for (int j : g[i]) if (j != l and !rem[j]) sz[i] +=
        dfs_sz(j, i);
191     return sz[i];
cbb }

85a int centroid(int i, int l, int size) {
994     for (int j : g[i]) if (j != l and !rem[j] and sz[j]
        > size / 2)
735         return centroid(j, i, size);
d9a     return i;
cbb }

d79 ll decomp(int i, int k) {
106     int c = centroid(i, i, dfs_sz(i));
a67     rem[c] = 1;

```

```

// gasta O(n) aqui - dfs sem ir pros caras removidos
04b     ll ans = 0;
020     vector<int> cnt(sz[i]);
878     cnt[0] = 1;
0a8     for (int j : g[c]) if (!rem[j]) {
5b4         vector<int> path;
baf         dfs(path, j);
1a1         for (int d : path) if (0 <= k-d-1 and k-d-1 <
            sz[i])
285             ans += cnt[k-d-1];
e8b         for (int d : path) cnt[d+1]++;
cbb     }

1c1     for (int j : g[c]) if (!rem[j]) ans += decomp(j, k);
3f1     rem[c] = 0;
ba7     return ans;
cbb }

```

## 2.8 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

845 vector<int> g[MAX], dist[MAX];
c1e int sz[MAX], rem[MAX], p[MAX];

071 int dfs_sz(int i, int l=-1) {
02c     sz[i] = 1;
e5c     for (int j : g[i]) if (j != l and !rem[j]) sz[i] +=
        dfs_sz(j, i);
191     return sz[i];
cbb }

85a int centroid(int i, int l, int size) {
994     for (int j : g[i]) if (j != l and !rem[j] and sz[j]
        > size / 2)
735         return centroid(j, i, size);

```

```

d9a     return i;
cbb }

324 void dfs_dist(int i, int l, int d=0) {
541     dist[i].push_back(d);
5a1     for (int j : g[i]) if (j != l and !rem[j])
82a         dfs_dist(j, i, d+1);
cbb }

27e void decomp(int i, int l = -1) {
106     int c = centroid(i, i, dfs_sz(i));
1b9     rem[c] = 1, p[c] = 1;
534     dfs_dist(c, c);
a2a     for (int j : g[c]) if (!rem[j]) decomp(j, c);
cbb }

76c void build(int n) {
235     for (int i = 0; i < n; i++) rem[i] = 0,
        dist[i].clear();
867     decomp(0);
96b     for (int i = 0; i < n; i++) reverse(dist[i].begin(),
        dist[i].end());
cbb }

```

## 2.9 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

eff ll d[MAX];
c0d vector<pair<int, int>> g[MAX]; // {vizinho, peso}

1a8 int n;

abc void dijkstra(int v) {
22c     for (int i = 0; i < n; i++) d[i] = LINF;
a7f     d[v] = 0;

```

```

88c     priority_queue<pair<ll, int>> pq;
b32     pq.emplace(0, v);

265     while (pq.size()) {
a25         auto [ndist, u] = pq.top(); pq.pop();
953         if (-ndist > d[u]) continue;

cda         for (auto [idx, w] : g[u]) if (d[idx] > d[u] +
w) {
331             d[idx] = d[u] + w;
a84             pq.emplace(-d[idx], idx);
cbb         }
cbb     }
cbb }

```

## 2.10 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
472 struct dinitz {
61f     const bool scaling = false; // com scaling -> O(nm
        log(MAXCAP)),
206         int lim; // com constante alta
670     struct edge {
358         int to, cap, rev, flow;
7f9         bool res;
d36         edge(int to_, int cap_, int rev_, bool res_)
a94             : to(to_), cap(cap_), rev(rev_), flow(0),
        res(res_) {}
214     };

002     vector<vector<edge>> g;
216     vector<int> lev, beg;
a71     ll F;
190     dinitz(int n) : g(n), F(0) {}

087     void add(int a, int b, int c) {
bae         g[a].emplace_back(b, c, g[b].size(), false);
4c6         g[b].emplace_back(a, 0, g[a].size()-1, true);

```

```

cbb    }
123    bool bfs(int s, int t) {
90f        lev = vector<int>(g.size(), -1); lev[s] = 0;
64c        beg = vector<int>(g.size(), 0);
8b2        queue<int> q; q.push(s);
402        while (q.size()) {
be1            int u = q.front(); q.pop();
bd9            for (auto& i : g[u]) {
dbc                if (lev[i.to] != -1 or (i.flow ==
i.cap)) continue;
b4f                if (scaling and i.cap - i.flow < lim)
continue;
185                lev[i.to] = lev[u] + 1;
8ca                q.push(i.to);
cbb            }
cbb        }
0de        return lev[t] != -1;
cbb    }
dfb    int dfs(int v, int s, int f = INF) {
50b        if (!f or v == s) return f;
88f        for (int& i = beg[v]; i < g[v].size(); i++) {
027            auto& e = g[v][i];
206            if (lev[e.to] != lev[v] + 1) continue;
ee0            int foi = dfs(e.to, s, min(f, e.cap -
e.flow));
749            if (!foi) continue;
3c5            e.flow += foi, g[e.to][e.rev].flow -= foi;
45c            return foi;
cbb        }
bb3        return 0;
cbb    }
ff6    ll max_flow(int s, int t) {
a86        for (lim = scaling ? (1<<30) : 1; lim; lim /= 2)
9d1            while (bfs(s, t)) while (int ff = dfs(s, t))
F += ff;
4ff        return F;
cbb    }
214 };

// Recupera as arestas do corte s-t
// d23977
dbd vector<pair<int, int>> get_cut(dinitz& g, int s, int t) {

```

```

f07        g.max_flow(s, t);
68c        vector<pair<int, int>> cut;
1b0        vector<int> vis(g.g.size(), 0), st = {s};
321        vis[s] = 1;
3c6        while (st.size()) {
b17            int u = st.back(); st.pop_back();
322            for (auto e : g.g[u]) if (!vis[e.to] and e.flow
< e.cap)
c17                vis[e.to] = 1, st.push_back(e.to);
cbb        }
481        for (int i = 0; i < g.g.size(); i++) for (auto e :
g.g[i])
9d2            if (vis[i] and !vis[e.to] and !e.res)
cut.emplace_back(i, e.to);
d1b        return cut;
cbb    }

```

## 2.11 Dominator Tree - Kawakami

```

// Se vira pra usar ai
//
// build - O(n)
// dominates - O(1)
// c80920

1a8 int n;

bbf namespace d_tree {
042     vector<int> g[MAX];

    // The dominator tree
b39     vector<int> tree[MAX];
5af     int dfs_l[MAX], dfs_r[MAX];

    // Auxiliary data
a2e     vector<int> rg[MAX], bucket[MAX];
3ef     int idom[MAX], sdom[MAX], prv[MAX], pre[MAX];
44b     int ancestor[MAX], label[MAX];
563     vector<int> preorder;

76a     void dfs(int v) {
6a1         static int t = 0;

```

```

db6         pre[v] = ++t;
767         sdom[v] = label[v] = v;
a3d         preorder.push_back(v);
d08         for (int nxt: g[v]) {
56c             if (sdom[nxt] == -1) {
eed                 prv[nxt] = v;
900                 dfs(nxt);
cbb             }
2b5             rg[nxt].push_back(v);
cbb         }
cbb     }
62e     int eval(int v) {
c93         if (ancestor[v] == -1) return v;
a75         if (ancestor[ancestor[v]] == -1) return label[v];
f33         int u = eval(ancestor[v]);
b49         if (pre[sdom[u]] < pre[sdom[label[v]]]) label[v]
= u;
66e         ancestor[v] = ancestor[u];
c24         return label[v];
cbb     }
4b2     void dfs2(int v) {
6a1         static int t = 0;
330         dfs_l[v] = t++;
5e0         for (int nxt: tree[v]) dfs2(nxt);
8e2         dfs_r[v] = t++;
cbb     }
c2c     void build(int s) {
603         for (int i = 0; i < n; i++) {
e6f             sdom[i] = pre[i] = ancestor[i] = -1;
2e1             rg[i].clear();
50a             tree[i].clear();
666             bucket[i].clear();
cbb         }
772         preorder.clear();
c6c         dfs(s);
12b         if (preorder.size() == 1) return;
3c7         for (int i = int(preorder.size()) - 1; i >= 1;
i--) {
6c6             int w = preorder[i];
a52             for (int v: rg[w]) {
5c1                 int u = eval(v);
a17                 if (pre[sdom[u]] < pre[sdom[w]]) sdom[w]

```

```

= sdom[u];
cbb         }
680         bucket[sdom[w]].push_back(w);
ea7         ancestor[w] = prv[w];
b99         for (int v: bucket[prv[w]]) {
5c1             int u = eval(v);
977             idom[v] = (u == v) ? sdom[v] : u;
cbb         }
2cc         bucket[prv[w]].clear();
cbb     }
d0c     for (int i = 1; i < preorder.size(); i++) {
6c6         int w = preorder[i];
14b         if (idom[w] != sdom[w]) idom[w] =
idom[idom[w]];
32f         tree[idom[w]].push_back(w);
cbb     }
8ac         idom[s] = sdom[s] = -1;
1b6         dfs2(s);
cbb     }

// Whether every path from s to v passes through u
490     bool dominates(int u, int v) {
c75         if (pre[v] == -1) return 1; // vacuously true
2ea         return dfs_l[u] <= dfs_l[v] && dfs_r[v] <=
dfs_r[u];
cbb     }
214 };

```

## 2.12 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]
// (vertice inicial nao repete)
//
// O(n+m)
// 7113df

63f template<bool directed=false> struct euler {
1a8     int n;
4c0     vector<vector<pair<int, int>>> g;
d63     vector<int> used;

30f     euler(int n_) : n(n_), g(n) {}
50f     void add(int a, int b) {
4cd         int at = used.size();
c51         used.push_back(0);
74e         g[a].emplace_back(b, at);
fab         if (!directed) g[b].emplace_back(a, at);
cbb     }
d41 #warning chamar para o src certo!
eed     pair<bool, vector<pair<int, int>>> get_path(int src)
{
baf         if (!used.size()) return {true, {}};
b25         vector<int> beg(n, 0);
4ec         for (int& i : used) i = 0;
// {{vertice, anterior}, label}
363         vector<pair<pair<int, int>, int>> ret, st =
{{{src, -1}, -1}};
3c6         while (st.size()) {
8ff             int at = st.back().first.first;
002             int& it = beg[at];
8a1             while (it < g[at].size() and
used[g[at][it].second]) it++;
8e4             if (it == g[at].size()) {
9dd                 if (ret.size() and
ret.back().first.second != at)
b82                     return {false, {}};
420                 ret.push_back(st.back()), st.pop_back();
9d9             } else {
daa                 st.push_back({g[at][it].first, at},
g[at][it].second});

```

```

eb8                 used[g[at][it].second] = 1;
cbb             }
cbb         }
a19         if (ret.size() != used.size()+1) return {false,
{}};
f77         vector<pair<int, int>> ans;
fdf         for (auto i : ret)
ans.emplace_back(i.first.first, i.second);
459         reverse(ans.begin(), ans.end());
997         return {true, ans};
cbb     }
9b6     pair<bool, vector<pair<int, int>>> get_cycle() {
baf         if (!used.size()) return {true, {}};
ad1         int src = 0;
34b         while (!g[src].size()) src++;
687         auto ans = get_path(src);
33c         if (!ans.first or ans.second[0].first !=
ans.second.back().first)
b82             return {false, {}};
350         ans.second[0].second = ans.second.back().second;
8b8         ans.second.pop_back();
ba7         return ans;
cbb     }
214 };

```

## 2.13 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

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

9f9 template<typename T> struct ETT {
    // treap
3c9     struct node {
ed1         node *l, *r, *p;
fa4         int pr, sz;
875         T val, sub, lazy;
53e         int id;
ffd         bool f; // se eh o 'first'
5ef         int qt_f; // numero de firsts na subarvore
7a8         node(int id_, T v, bool f_ = 0) : l(NULL),
r(NULL), p(NULL), pr(rng()),
62b         sz(1), val(v), sub(v), lazy(), id(id_),
f(f_), qt_f(qt_f) {}
a9c         void prop() {
d09             if (lazy != T()) {
021                 if (f) val += lazy;
971                 sub += lazy*sz;
b87                 if (l) l->lazy += lazy;
d3b                 if (r) r->lazy += lazy;
cbb             }
bfd             lazy = T();
cbb         }
01e         void update() {
8da             sz = 1, sub = val, qt_f = f;
171             if (l) l->prop(), sz += l->sz, sub +=
l->sub, qt_f += l->qt_f;
117             if (r) r->prop(), sz += r->sz, sub +=
r->sub, qt_f += r->qt_f;
cbb         }
214     };

bb7     node* root;

```

```

73c     int size(node* x) { return x ? x->sz : 0; }
bcf     void join(node* l, node* r, node*& i) { // assume
    que l < r
986         if (!l or !r) return void(i = l ? l : r);
161         l->prop(), r->prop();
ff5         if (l->pr > r->pr) join(l->r, r, l->r), l->r->p
= i = l;
982         else join(l, r->l, r->l), r->l->p = i = r;
bda         i->update();
cbb     }
a20     void split(node* i, node*& l, node*& r, int v, int
key = 0) {
26a         if (!i) return void(r = l = NULL);
c89         i->prop();
d9e         if (key + size(i->l) < v) {
448             split(i->r, i->r, r, v, key+size(i->l)+1), l
= i;
a21             if (r) r->p = NULL;
6e8             if (i->r) i->r->p = i;
9d9         } else {
98d             split(i->l, l, i->l, v, key), r = i;
5a3             if (l) l->p = NULL;
899             if (i->l) i->l->p = i;
cbb         }
bda         i->update();
cbb     }
ac7     int get_idx(node* i) {
6cf         int ret = size(i->l);
482         for (; i->p; i = i->p) {
fbf             node* pai = i->p;
8a6             if (i != pai->l) ret += size(pai->l) + 1;
cbb         }
edf         return ret;
cbb     }
048     node* get_min(node* i) {
433         if (!i) return NULL;
f8e         return i->l ? get_min(i->l) : i;
cbb     }
f03     node* get_max(node* i) {
433         if (!i) return NULL;
424         return i->r ? get_max(i->r) : i;
cbb     }

```

```

// fim da treap

4fb    vector<node*> first, last;

f82    ETT(int n, vector<T> v = {}) : root(NULL), first(n),
last(n) {
c5e        if (!v.size()) v = vector<T>(n);
603        for (int i = 0; i < n; i++) {
a00            first[i] = last[i] = new node(i, v[i], 1);
469            join(root, first[i], root);
cbb        }
cbb    }
83f    ETT(const ETT& t) { throw logic_error("Nao copiar a
ETT!"); }
c09    ~ETT() {
609        vector<node*> q = {root};
402        while (q.size()) {
e5d            node* x = q.back(); q.pop_back();
ee9            if (!x) continue;
1c7            q.push_back(x->l), q.push_back(x->r);
bf0            delete x;
cbb        }
cbb    }

153    pair<int, int> get_range(int i) {
670        return {get_idx(first[i]), get_idx(last[i])};
cbb    }
7af    void link(int v, int u) { // 'v' tem que ser raiz
890        auto [lv, rv] = get_range(v);
f13        int ru = get_idx(last[u]);

4b4        node* V;
df9        node *L, *M, *R;
117        split(root, M, R, rv+1), split(M, L, M, lv);
f1e        V = M;
a28        join(L, R, root);

e66        split(root, L, R, ru+1);
367        join(L, V, L);
7e8        join(L, last[u] = new node(u, T() /* elemento
neutro */, L);
a28        join(L, R, root);

```

```

cbb    }
4e6    void cut(int v) {
892        auto [l, r] = get_range(v);

df9        node *L, *M, *R;
dca        split(root, M, R, r+1), split(M, L, M, l);
de6        node *LL = get_max(L), *RR = get_min(R);
710        if (LL and RR and LL->id == RR->id) { // remove
duplicata
e8b            if (last[RR->id] == RR) last[RR->id] = LL;
992            node *A, *B;
6b3            split(R, A, B, 1);
10c            delete A;
9d5            R = B;
cbb        }
a28        join(L, R, root);
a0d        join(root, M, root);
cbb    }
808    T query(int v) {
892        auto [l, r] = get_range(v);
df9        node *L, *M, *R;
dca        split(root, M, R, r+1), split(M, L, M, l);
d43        T ans = M->sub;
69d        join(L, M, M), join(M, R, root);
ba7        return ans;
cbb    }
93b    void update(int v, T val) { // soma val em todo
        mundo da subarvore
892        auto [l, r] = get_range(v);
df9        node *L, *M, *R;
dca        split(root, M, R, r+1), split(M, L, M, l);
409        M->lazy += val;
69d        join(L, M, M), join(M, R, root);
cbb    }
129    void update_v(int v, T val) { // muda o valor de v
        pra val
ac1        int l = get_idx(first[v]);
df9        node *L, *M, *R;
d0c        split(root, M, R, l+1), split(M, L, M, l);
25e        M->val = M->sub = val;
69d        join(L, M, M), join(M, R, root);
cbb    }

```

```

934     bool is_in_subtree(int v, int u) { // se u ta na
      subtree de v
990         auto [lv, rv] = get_range(v);
6ec         auto [lu, ru] = get_range(u);
732         return lv <= lu and ru <= rv;
cbb     }

355     void print(node* i) {
eae         if (!i) return;
a1e         print(i->l);
743         cout << i->id+1 << " ";
f15         print(i->r);
cbb     }
065     void print() { print(root); cout << endl; }
214 };

```

## 2.14 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

1a8 int n;
ae5 int d[MAX][MAX];

73c bool floyd_warshall() {
e22     for (int k = 0; k < n; k++)
830     for (int i = 0; i < n; i++)
f90     for (int j = 0; j < n; j++)
0ab         d[i][j] = min(d[i][j], d[i][k] + d[k][j]);

830     for (int i = 0; i < n; i++)
753         if (d[i][i] < 0) return 1;

bb3     return 0;
cbb }

```

## 2.15 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)))
// 51fabe

6ef namespace func_graph {
1a8     int n;
ce2     int f[MAX], vis[MAX], d[MAX];
f82     int p[MAX], pp[MAX], rt[MAX], pos[MAX];
ebd     int sz[MAX], comp;
6a9     vector<vector<int>> ciclo;
405     ll val[MAX], jmp[MAX], seg[2*MAX];

97c     ll op(ll a, ll b) { return a+b; }; // mudar a
operacao aqui
27b     void dfs(int i, int t = 2) {
9c9         vis[i] = t;
f09         if (vis[f[i]] >= 2) { // comeca ciclo - f[i] eh
o rep.
e0a             d[i] = 0, rt[i] = comp;
74c             sz[comp] = t - vis[f[i]] + 1;
97b             p[i] = pp[i] = i, jmp[i] = val[i];
15c             ciclo.emplace_back();
bfb             ciclo.back().push_back(i);
9d9         } else {
c16             if (!vis[f[i]]) dfs(f[i], t+1);
8c0             rt[i] = rt[f[i]];
195             if (sz[comp]+1) { // to no ciclo

```

```

d0f          d[i] = 0;
97b          p[i] = pp[i] = i, jmp[i] = val[i];
bfb          ciclo.back().push_back(i);
9d9      } else { // nao to no ciclo
00d          d[i] = d[f[i]]+1, p[i] = f[i];
511          pp[i] = 2*d[pp[f[i]]] ==
d[pp[pp[f[i]]]]+d[f[i]] ? pp[pp[f[i]]] : f[i];
114          jmp[i] = pp[i] == f[i] ? val[i] :
op(val[i], op(jmp[f[i]], jmp[pp[f[i]]]));
cbb      }
cbb      }
e4a      if (f[ciclo[rt[i]][0]] == i) comp++; // fim do
ciclo
29a      vis[i] = 1;
cbb      }
1da      void build(vector<int> f_, vector<int> val_ = {}) {
bcb          n = f_.size(), comp = 0;
527          if (!val_.size()) val_ = f_;
830          for (int i = 0; i < n; i++)
998              f[i] = f_[i], val[i] = val_[i], vis[i] = 0,
sz[i] = -1;

e74      ciclo.clear();
158      for (int i = 0; i < n; i++) if (!vis[i]) dfs(i);
6bb      int t = 0;
daa      for (auto& c : ciclo) {
336          reverse(c.begin(), c.end());
ea5          for (int j : c) {
85b              pos[j] = t;
948              seg[n+t] = val[j];
c82              t++;
cbb          }
cbb      }
dc1      for (int i = n-1; i; i--) seg[i] = op(seg[2*i],
seg[2*i+1]);
cbb      }

283      int f_k(int i, ll k) {
1b1          while (d[i] and k) {
77b              int big = d[i] - d[pp[i]];
ded              if (big <= k) k -= big, i = pp[i];
584              else k--, i = p[i];

```

```

cbb      }
77e          if (!k) return i;
a19          return ciclo[rt[i]][(pos[i] -
pos[ciclo[rt[i]][0]] + k) % sz[rt[i]]];
cbb      }
047      ll path(int i, ll k) {
3cf          auto query = [&](int l, int r) {
3e4              ll q = 0;
47a              for (l += n, r += n; l <= r; ++l/=2, --r/=2)
{
27e                  if (l%2 == 1) q = op(q, seg[l]);
1f2                  if (r%2 == 0) q = op(q, seg[r]);
cbb              }
bef              return q;
214          };
b73          ll ret = 0;
1b1          while (d[i] and k) {
77b              int big = d[i] - d[pp[i]];
327              if (big <= k) k -= big, ret = op(ret,
jmp[i]), i = pp[i];
f9e              else k--, ret = op(ret, val[i]), i = p[i];
cbb          }
e3c          if (!k) return ret;
a9e          int first = pos[ciclo[rt[i]][0]], last =
pos[ciclo[rt[i]].back()];

// k/sz[rt[i]] voltas completas
430          if (k/sz[rt[i]]) ret = op(ret, k/sz[rt[i]] *
query(first, last));

9af          k %= sz[rt[i]];
e3c          if (!k) return ret;
8ea          int l = pos[i], r = first + (pos[i] - first + k
- 1) % sz[rt[i]];
982          if (l <= r) return op(ret, query(l, r));
687          return op(ret, op(query(l, last), query(first,
r)));
cbb      }
cbb }

```

## 2.16 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))

556 namespace seg { ... }

// 599946
826 namespace hld {
c0d     vector<pair<int, int> > g[MAX];
e65     int pos[MAX], sz[MAX];
7c0     int sobe[MAX], pai[MAX];
096     int h[MAX], v[MAX], t;

0ce     void build_hld(int k, int p = -1, int f = 1) {
180         v[pos[k] = t++] = sobe[k]; sz[k] = 1;
418         for (auto& i : g[k]) if (i.first != p) {
dd2             auto [u, w] = i;
a76             sobe[u] = w; pai[u] = k;
0c1             h[u] = (i == g[k][0] ? h[k] : u);
da7             build_hld(u, k, f); sz[k] += sz[u];

865             if (sz[u] > sz[g[k][0].first] or
g[k][0].first == p)
9a3                 swap(i, g[k][0]);
cbb         }
667         if (p*f == -1) build_hld(h[k] = k, -1, t = 0);
cbb     }
1f8     void build(int root = 0) {
a34         t = 0;
295         build_hld(root);
c83         seg::build(t, v);
cbb     }
3fc     ll query_path(int a, int b) {
2d5         if (a == b) return 0;
aa1         if (pos[a] < pos[b]) swap(a, b);

```

```

29b         if (h[a] == h[b]) return seg::query(pos[b]+1,
pos[a]);
fca         return seg::query(pos[h[a]], pos[a]) +
query_path(pai[h[a]], b);
cbb     }
920     void update_path(int a, int b, int x) {
d54         if (a == b) return;
aa1         if (pos[a] < pos[b]) swap(a, b);

881         if (h[a] == h[b]) return
(void)seg::update(pos[b]+1, pos[a], x);
701         seg::update(pos[h[a]], pos[a], x);
update_path(pai[h[a]], b, x);
cbb     }
d0a     ll query_subtree(int a) {
b9f         if (sz[a] == 1) return 0;
2f6         return seg::query(pos[a]+1, pos[a]+sz[a]-1);
cbb     }
acc     void update_subtree(int a, int x) {
a5a         if (sz[a] == 1) return;
9cd         seg::update(pos[a]+1, pos[a]+sz[a]-1, x);
cbb     }
7be     int lca(int a, int b) {
aa1         if (pos[a] < pos[b]) swap(a, b);
ca5         return h[a] == h[b] ? b : lca(pai[h[a]], b);
cbb     }
cbb }

```

## 2.17 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))

556 namespace seg { ... }

```

```

// de3d84
826 namespace hld {
042     vector<int> g[MAX];
e65     int pos[MAX], sz[MAX];
bd4     int peso[MAX], pai[MAX];
096     int h[MAX], v[MAX], t;

0ce     void build_hld(int k, int p = -1, int f = 1) {
b18         v[pos[k] = t++] = peso[k]; sz[k] = 1;
b94         for (auto& i : g[k]) if (i != p) {
78d             pai[i] = k;
26e             h[i] = (i == g[k][0] ? h[k] : i);
193             build_hld(i, k, f); sz[k] += sz[i];

cd1             if (sz[i] > sz[g[k][0]] or g[k][0] == p)
swap(i, g[k][0]);
cbb         }
667         if (p*f == -1) build_hld(h[k] = k, -1, t = 0);
cbb     }
1f8     void build(int root = 0) {
a34         t = 0;
295         build_hld(root);
c83         seg::build(t, v);
cbb     }
3fc     ll query_path(int a, int b) {
aa1         if (pos[a] < pos[b]) swap(a, b);

4bf         if (h[a] == h[b]) return seg::query(pos[b],
pos[a]);
fca         return seg::query(pos[h[a]], pos[a]) +
query_path(pai[h[a]], b);
cbb     }
920     void update_path(int a, int b, int x) {
aa1         if (pos[a] < pos[b]) swap(a, b);

198         if (h[a] == h[b]) return
(void)seg::update(pos[b], pos[a], x);
701         seg::update(pos[h[a]], pos[a], x);
update_path(pai[h[a]], b, x);
cbb     }
d0a     ll query_subtree(int a) {
b3e         return seg::query(pos[a], pos[a]+sz[a]-1);

```

```

cbb     }
acc     void update_subtree(int a, int x) {
a22         seg::update(pos[a], pos[a]+sz[a]-1, x);
cbb     }
7be     int lca(int a, int b) {
aa1         if (pos[a] < pos[b]) swap(a, b);
ca5         return h[a] == h[b] ? b : lca(pai[h[a]], b);
cbb     }
cbb }

```

## 2.18 Heavy-Light Decomposition sem Update

```

// query de min do caminho
//
// Complexidades:
// build - O(n)
// query_path - O(log(n))
// ee6991

```

```

826 namespace hld {
c0d     vector<pair<int, int> > g[MAX];
e65     int pos[MAX], sz[MAX];
7c0     int sobe[MAX], pai[MAX];
096     int h[MAX], v[MAX], t;
ea2     int men[MAX], seg[2*MAX];

0ce     void build_hld(int k, int p = -1, int f = 1) {
180         v[pos[k] = t++] = sobe[k]; sz[k] = 1;
418         for (auto& i : g[k]) if (i.first != p) {
1f5             sobe[i.first] = i.second; pai[i.first] = k;
6fa             h[i.first] = (i == g[k][0] ? h[k] : i.first);
87b             men[i.first] = (i == g[k][0] ? min(men[k],
i.second) : i.second);
4b2             build_hld(i.first, k, f); sz[k] +=
sz[i.first];

bc3             if (sz[i.first] > sz[g[k][0].first] or
g[k][0].first == p)
9a3                 swap(i, g[k][0]);
cbb             }
667         if (p*f == -1) build_hld(h[k] = k, -1, t = 0);
cbb     }

```

```

1f8     void build(int root = 0) {
a34         t = 0;
295         build_hld(root);
3ae         for (int i = 0; i < t; i++) seg[i+t] = v[i];
8db         for (int i = t-1; i; i--) seg[i] = min(seg[2*i],
            seg[2*i+1]);
cbb     }
f04     int query_path(int a, int b) {
490         if (a == b) return INF;
aa1         if (pos[a] < pos[b]) swap(a, b);

98f         if (h[a] != h[b]) return min(men[a],
            query_path(pai[h[a]], b));
46b         int ans = INF, x = pos[b]+1+t, y = pos[a]+t;
646         for (; x <= y; ++x/=2, --y/=2) ans = min({ans,
            seg[x], seg[y]});
ba7         return ans;
cbb     }
214 };

```

## 2.19 Isomorfismo de arvores

```

// thash() retorna o hash da arvore (usando centroids como
// vertices especiais).
// Duas arvores sao isomorfas sse seu hash eh o mesmo
//
// 0(|V|.log(|V|))
// 8fb6bb

```

```

91f map<vector<int>, int> mhash;

df6 struct tree {
1a8     int n;
789     vector<vector<int>> g;
347     vector<int> sz, cs;

1b5     tree(int n_) : n(n_), g(n_), sz(n_) {}

76b     void dfs_centroid(int v, int p) {
588         sz[v] = 1;
fa7         bool cent = true;
18e         for (int u : g[v]) if (u != p) {

```

```

365             dfs_centroid(u, v), sz[v] += sz[u];
e90             if (sz[u] > n/2) cent = false;
cbb         }
1f6         if (cent and n - sz[v] <= n/2) cs.push_back(v);
cbb     }
784     int fhash(int v, int p) {
544         vector<int> h;
332         for (int u : g[v]) if (u != p)
            h.push_back(fhash(u, v));
1c9         sort(h.begin(), h.end());
3ac         if (!mhash.count(h)) mhash[h] = mhash.size();
bbc         return mhash[h];
cbb     }
38f     ll thash() {
23a         cs.clear();
3a5         dfs_centroid(0, -1);
16d         if (cs.size() == 1) return fhash(cs[0], -1);
772         ll h1 = fhash(cs[0], cs[1]), h2 = fhash(cs[1],
            cs[0]);
fae         return (min(h1, h2) << 30) + max(h1, h2);
cbb     }
214 };

```

## 2.20 Kosaraju

```

// 0(n + m)
// a4f310

1a8 int n;
042 vector<int> g[MAX];
58d vector<int> gi[MAX]; // grafo invertido
c5a int vis[MAX];
ee6 stack<int> S;
a52 int comp[MAX]; // componente conexo de cada vertice

1ca void dfs(int k) {
59a     vis[k] = 1;
54f     for (int i = 0; i < (int) g[k].size(); i++)
8d5         if (!vis[g[k][i]]) dfs(g[k][i]);

58f     S.push(k);
cbb }

```

```

436 void scc(int k, int c) {
59a     vis[k] = 1;
52c     comp[k] = c;
ff0     for (int i = 0; i < (int) gi[k].size(); i++)
bf6         if (!vis[gi[k][i]]) scc(gi[k][i], c);
cbb }

db8 void kosaraju() {
991     for (int i = 0; i < n; i++) vis[i] = 0;
158     for (int i = 0; i < n; i++) if (!vis[i]) dfs(i);

991     for (int i = 0; i < n; i++) vis[i] = 0;
d32     while (S.size()) {
70b         int u = S.top();
7de         S.pop();
f43         if (!vis[u]) scc(u, u);
cbb     }
cbb }

```

## 2.21 Kruskal

```

// Gera e retorna uma AGM e seu custo total a partir do
// vetor de arestas (edg)
// do grafo
//
// O(m log(m) + m a(m))
// 864875

1b9 vector<tuple<int, int, int>> edg; // {peso, [x,y]}

// DSU em O(a(n))
4a6 void dsu_build();
d78 int find(int a);
369 void unite(int a, int b);

c67 pair<ll, vector<tuple<int, int, int>>> kruskal(int n) {
8d2     dsu_build(n);
e31     sort(edg.begin(), edg.end());

854     ll cost = 0;
979     vector<tuple<int, int, int>> mst;

```

```

fea     for (auto [w,x,y] : edg) if (find(x) != find(y)) {
9de         mst.emplace_back(w, x, y);
45f         cost += w;
05a         unite(x,y);
cbb     }
5df     return {cost, mst};
cbb }

```

## 2.22 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

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

// b0dda3
6c6 struct kuhn {
14e     int n, m;
789     vector<vector<int>>> g;
d3f     vector<int> vis, ma, mb;

40e     kuhn(int n_, int m_) : n(n_), m(m_), g(n),
8af         vis(n+m), ma(n, -1), mb(m, -1) {}

ba6     void add(int a, int b) { g[a].push_back(b); }

caf     bool dfs(int i) {
29a         vis[i] = 1;
29b         for (int j : g[i]) if (!vis[n+j]) {
8c9             vis[n+j] = 1;
2cf             if (mb[j] == -1 or dfs(mb[j])) {
bfe                 ma[i] = j, mb[j] = i;

```



```

8a6         return true;
cbb     }
cbb     }
dif     return false;
cbb     }
bf7     int matching() {
1ae         int ret = 0, aum = 1;
5a8         for (auto& i : g) shuffle(i.begin(), i.end(),
rng);
392         while (aum) {
618             for (int j = 0; j < m; j++) vis[n+j] = 0;
c5d             aum = 0;
830             for (int i = 0; i < n; i++)
01f                 if (ma[i] == -1 and dfs(i)) ret++, aum =
1;
cbb         }
edf         return ret;
cbb     }
214 };

// 55fb67
ebf pair<vector<int>, vector<int>> recover(kuhn& K) {
e80     K.matching();
50c     int n = K.n, m = K.m;
9d0     for (int i = 0; i < n+m; i++) K.vis[i] = 0;
bde     for (int i = 0; i < n; i++) if (K.ma[i] == -1)
K.dfs(i);
8ad     vector<int> ca, cb;
576     for (int i = 0; i < n; i++) if (!K.vis[i])
ca.push_back(i);
f24     for (int i = 0; i < m; i++) if (K.vis[n+i])
cb.push_back(i);
aad     return {ca, cb};
cbb }

```

## 2.23 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))

677 vector<vector<int> > g(MAX);
41c int n, p;
e75 int pai[MAX2][MAX];
999 int in[MAX], out[MAX];

1ca void dfs(int k) {
fdf     in[k] = p++;
54f     for (int i = 0; i < (int) g[k].size(); i++)
9b7         if (in[g[k][i]] == -1) {
ba6             pai[0][g[k][i]] = k;
c38             dfs(g[k][i]);
cbb         }
26f     out[k] = p++;
cbb }

c11 void build(int raiz) {
a67     for (int i = 0; i < n; i++) pai[0][i] = i;
c63     p = 0, memset(in, -1, sizeof in);
ecb     dfs(raiz);

// pd dos pais
511     for (int k = 1; k < MAX2; k++) for (int i = 0; i <
n; i++)
d38         pai[k][i] = pai[k - 1][pai[k - 1][i]];
cbb }

00f bool anc(int a, int b) { // se a eh ancestral de b
bfe     return in[a] <= in[b] and out[a] >= out[b];
cbb }

7be int lca(int a, int b) {
86d     if (anc(a, b)) return a;
e52     if (anc(b, a)) return b;

// sobe a
f70     for (int k = MAX2 - 1; k >= 0; k--)
acf         if (!anc(pai[k][a], b)) a = pai[k][a];

847     return pai[0][a];

```

```

cbb }

// 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))

9c6 int d[MAX], p[MAX], pp[MAX];

d40 void set_root(int i) { p[i] = pp[i] = i, d[i] = 0; }

e9d void add_leaf(int i, int u) {
e0b     p[i] = u, d[i] = d[u]+1;
b15     pp[i] = 2*d[pp[u]] == d[pp[pp[u]]]+d[u] ? pp[pp[u]]
: u;
cbb }

c37 int kth(int i, int k) {
4e3     int dd = max(0, d[i]-k);
935     while (d[i] > dd) i = d[pp[i]] >= dd ? pp[i] : p[i];
d9a     return i;
cbb }

7be int lca(int a, int b) {
a69     if (d[a] < d[b]) swap(a, b);
6cd     while (d[a] > d[b]) a = d[pp[a]] >= d[b] ? pp[a] :
p[a];
984     while (a != b) {
932         if (pp[a] != pp[b]) a = pp[a], b = pp[b];
e7c         else a = p[a], b = p[b];
cbb     }
3f5     return a;
cbb }

4fe int dist(int a, int b) { return d[a]+d[b]-2*d[lca(a,b)];
}

042 vector<int> g[MAX];

```

```

3ab void build(int i, int pai=-1) {
5cf     if (pai == -1) set_root(i);
15f     for (int j : g[i]) if (j != pai) {
d31         add_leaf(j, i);
b21         build(j, i);
cbb     }
cbb }

```

## 2.24 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

042 vector<int> g[MAX];
713 int pos[MAX], h[MAX], sz[MAX];
ff1 int pai[MAX], t;

8bf void build(int k, int p = -1, int f = 1) {
bce     pos[k] = t++; sz[k] = 1;
e26     for (int& i : g[k]) if (i != p) {
78d         pai[i] = k;
26e         h[i] = (i == g[k][0] ? h[k] : i);
cb8         build(i, k, f); sz[k] += sz[i];

cd1         if (sz[i] > sz[g[k][0]] or g[k][0] == p) swap(i,
g[k][0]);
cbb     }
3da     if (p*f == -1) t = 0, h[k] = k, build(k, -1, 0);
cbb }

7be int lca(int a, int b) {
aa1     if (pos[a] < pos[b]) swap(a, b);
ca5     return h[a] == h[b] ? b : lca(pai[h[a]], b);
cbb }

```

```

00f bool anc(int a, int b) {
db5     return pos[a] <= pos[b] and pos[b] <= pos[a]+sz[a]-1;
cbb }

```

## 2.25 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
1a5 template<typename T> struct rmq {
517     vector<T> v;
fcc     int n; static const int b = 30;
70e     vector<int> mask, t;

18e     int op(int x, int y) { return v[x] < v[y] ? x : y; }
ee1     int msb(int x) { return
        __builtin_clz(1)-__builtin_clz(x); }
6ad     rmq() {}
43c     rmq(const vector<T>& v_) : v(v_), n(v.size()),
        mask(n), t(n) {
2e5         for (int i = 0, at = 0; i < n; mask[i++] = at |=
1) {
a61             at = (at<<1)&((1<<b)-1);
76a             while (at and op(i, i-msb(at&-at)) == i) at
                ^= at&-at;
cbb         }
243         for (int i = 0; i < n/b; i++) t[i] =
            b*i+b-1-msb(mask[b*i+b-1]);
39d         for (int j = 1; (1<<j) <= n/b; j++) for (int i =
0; i+(1<<j) <= n/b; i++)
ba5             t[n/b*j+i] = op(t[n/b*(j-1)+i],
                t[n/b*(j-1)+i+(1<<(j-1))]);
cbb     }

```

```

c92     int small(int r, int sz = b) { return
r-msb(mask[r]&((1<<sz)-1)); }
b7a     T query(int l, int r) {
27b         if (r-l+1 <= b) return small(r, r-l+1);
7bf         int ans = op(small(l+b-1), small(r));
e80         int x = l/b+1, y = r/b-1;
e25         if (x <= y) {
a4e             int j = msb(y-x+1);
002             ans = op(ans, op(t[n/b*j+x],
                t[n/b*j+y-(1<<j)+1]));
cbb         }
ba7         return ans;
cbb     }
214 };

// 645120
065 namespace lca {
042     vector<int> g[MAX];
8ec     int v[2*MAX], pos[MAX], dep[2*MAX];
8bd     int t;
2de     rmq<int> RMQ;

4cf     void dfs(int i, int d = 0, int p = -1) {
c97         v[t] = i, pos[i] = t, dep[t++] = d;
cac         for (int j : g[i]) if (j != p) {
8ec             dfs(j, d+1, i);
cf2             v[t] = i, dep[t++] = d;
cbb         }
cbb     }
789     void build(int n, int root) {
a34         t = 0;
14e         dfs(root);
3f4         RMQ = rmq<int>(vector<int>(dep, dep+2*n-1));
cbb     }
7be     int lca(int a, int b) {
ab7         a = pos[a], b = pos[b];
9c0         return v[RMQ.query(min(a, b), max(a, b))];
cbb     }
b5d     int dist(int a, int b) {
670         return dep[pos[a]] + dep[pos[b]] -
            2*dep[pos[lca(a, b)]];
cbb     }

```

```
cbb }
```

## 2.26 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

1a8 int n;

3ae namespace linetree {
f37     int id[MAX], seg[2*MAX], pos[MAX];
43f     vector<int> v[MAX], val[MAX];
430     vector<pair<int, pair<int, int> > > ar;

dc6     void add(int a, int b, int p) { ar.push_back({p, {a,
b}}}); }
0a8     void build() {
b09         sort(ar.rbegin(), ar.rend());
0e3         for (int i = 0; i < n; i++) id[i] = i, v[i] =
{i}, val[i].clear();
8bb         for (auto i : ar) {
c91             int a = id[i.second.first], b =
id[i.second.second];
f6f             if (a == b) continue;
c58             if (v[a].size() < v[b].size()) swap(a, b);
fb8             for (auto j : v[b]) id[j] = a,
v[a].push_back(j);
482             val[a].push_back(i.first);
78b             for (auto j : val[b]) val[a].push_back(j);
e39             v[b].clear(), val[b].clear();
cbb         }
8e8         vector<int> vv;
2ce         for (int i = 0; i < n; i++) for (int j = 0; j <
v[i].size(); j++) {
e52             pos[v[i][j]] = vv.size();
941             if (j + 1 < v[i].size())
```

```
vv.push_back(val[i][j]);
1cb         else vv.push_back(0);
cbb     }
bb4     for (int i = n; i < 2*n; i++) seg[i] = vv[i-n];
69e     for (int i = n-1; i; i--) seg[i] = min(seg[2*i],
seg[2*i+1]);
cbb     }
4ea     int query(int a, int b) {
596         if (id[a] != id[b]) return 0; // nao estao
conectados
ab7         a = pos[a], b = pos[b];
d11         if (a > b) swap(a, b);
199         b--;
38a         int ans = INF;
513         for (a += n, b += n; a <= b; ++a/=2, --b/=2) ans
= min({ans, seg[a], seg[b]});
ba7         return ans;
cbb     }
214 };
```

## 2.27 Link-cut Tree

```
// Link-cut tree padrao
//
// Todas as operacoes sao O(log(n)) amortizado
// e4e663

1ef namespace lct {
3c9     struct node {
19f         int p, ch[2];
062         node() { p = ch[0] = ch[1] = -1; }
214     };

5f3     node t[MAX];

971     bool is_root(int x) {
657         return t[x].p == -1 or (t[t[x].p].ch[0] != x and
t[t[x].p].ch[1] != x);
cbb     }
ed6     void rotate(int x) {
497         int p = t[x].p, pp = t[p].p;
fc4         if (!is_root(p)) t[pp].ch[t[pp].ch[1] == p] = x;
```

```

251         bool d = t[p].ch[0] == x;
461         t[p].ch[!d] = t[x].ch[d], t[x].ch[d] = p;
a76         if (t[p].ch[!d]+1) t[t[p].ch[!d]].p = p;
8fa         t[x].p = pp, t[p].p = x;
cbb     }
07c     void splay(int x) {
18c         while (!is_root(x)) {
497             int p = t[x].p, pp = t[p].p;
0c5             if (!is_root(p)) rotate((t[pp].ch[0] ==
p)^(t[p].ch[0] == x) ? x : p);
64f             rotate(x);
cbb         }
cbb     }
f16     int access(int v) {
0eb         int last = -1;
01a         for (int w = v; w+1; last = w, splay(v), w =
t[v].p)
024             splay(w), t[w].ch[1] = (last == -1 ? -1 : v);
3d3         return last;
cbb     }
e89     int find_root(int v) {
5e3         access(v);
3de         while (t[v].ch[0]+1) v = t[v].ch[0];
f05         return splay(v), v;
cbb     }
142     void link(int v, int w) { // v deve ser raiz
5e3         access(v);
10d         t[v].p = w;
cbb     }
4e6     void cut(int v) { // remove aresta de v pro pai
5e3         access(v);
264         t[v].ch[0] = t[t[v].ch[0]].p = -1;
cbb     }
bbb     int lca(int v, int w) {
948         return access(v), access(w);
cbb     }
cbb }

```

## 2.28 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

1ef namespace lct {
3c9     struct node {
19f         int p, ch[2];
810         ll val, sub;
aa6         bool rev;
04a         int sz, ar;
4e4         ll lazy;
f93         node() {}
7a8         node(int v, int ar_) :
546             p(-1), val(v), sub(v), rev(0), sz(ar_), ar(ar_),
lazy(0) {
b07             ch[0] = ch[1] = -1;
cbb         }
214     };

c53     node t[2*MAX]; // MAXN + MAXQ
99e     map<pair<int, int>, int> aresta;
e4d     int sz;

95a     void prop(int x) {
dc1         if (t[x].lazy) {
25e             if (t[x].ar) t[x].val += t[x].lazy;
2ab             t[x].sub += t[x].lazy*t[x].sz;
edc             if (t[x].ch[0]+1) t[t[x].ch[0]].lazy +=
t[x].lazy;
942             if (t[x].ch[1]+1) t[t[x].ch[1]].lazy +=
t[x].lazy;
cbb         }
aa2         if (t[x].rev) {
f95             swap(t[x].ch[0], t[x].ch[1]);
379             if (t[x].ch[0]+1) t[t[x].ch[0]].rev ^= 1;
c3d             if (t[x].ch[1]+1) t[t[x].ch[1]].rev ^= 1;
cbb         }
230         t[x].lazy = 0, t[x].rev = 0;
cbb     }
564     void update(int x) {

```

```

1a3      t[x].sz = t[x].ar, t[x].sub = t[x].val;
8ca      for (int i = 0; i < 2; i++) if (t[x].ch[i]+1) {
621          prop(t[x].ch[i]);
c4f          t[x].sz += t[t[x].ch[i]].sz;
269          t[x].sub += t[t[x].ch[i]].sub;
cbb      }
cbb      }
971      bool is_root(int x) {
657          return t[x].p == -1 or (t[t[x].p].ch[0] != x and
t[t[x].p].ch[1] != x);
cbb      }
ed6      void rotate(int x) {
497          int p = t[x].p, pp = t[p].p;
fc4          if (!is_root(p)) t[pp].ch[t[pp].ch[1] == p] = x;
251          bool d = t[p].ch[0] == x;
461          t[p].ch[!d] = t[x].ch[d], t[x].ch[d] = p;
a76          if (t[p].ch[!d]+1) t[t[p].ch[!d]].p = p;
8fa          t[x].p = pp, t[p].p = x;
444          update(p), update(x);
cbb      }
238      int splay(int x) {
18c          while (!is_root(x)) {
497              int p = t[x].p, pp = t[p].p;
77b              if (!is_root(p)) prop(pp);
be5              prop(p), prop(x);
0c5              if (!is_root(p)) rotate((t[pp].ch[0] ==
p)^(t[p].ch[0] == x) ? x : p);
64f              rotate(x);
cbb          }
aab          return prop(x), x;
cbb      }
f16      int access(int v) {
0eb          int last = -1;
d9f          for (int w = v; w+1; update(last = w), splay(v),
w = t[v].p)
024              splay(w), t[w].ch[1] = (last == -1 ? -1 : v);
3d3          return last;
cbb      }
9f1      void make_tree(int v, int w=0, int ar=0) { t[v] =
node(w, ar); }
e89      int find_root(int v) {
13f          access(v), prop(v);

```

```

9f0          while (t[v].ch[0]+1) v = t[v].ch[0], prop(v);
637          return splay(v);
cbb      }
82f      bool conn(int v, int w) {
2cf          access(v), access(w);
b9b          return v == w ? true : t[v].p != -1;
cbb      }
277      void rootify(int v) {
5e3          access(v);
a02          t[v].rev ^= 1;
cbb      }
971      ll query(int v, int w) {
b54          rootify(w), access(v);
249          return t[v].sub;
cbb      }
3fa      void update(int v, int w, int x) {
b54          rootify(w), access(v);
12c          t[v].lazy += x;
cbb      }
204      void link_(int v, int w) {
821          rootify(w);
389          t[w].p = v;
cbb      }
6b8      void link(int v, int w, int x) { // v--w com peso x
379          int id = MAX + sz++;
110          aresta[make_pair(v, w)] = id;
a88          make_tree(id, x, 1);
c88          link_(v, id), link_(id, w);
cbb      }
e63      void cut_(int v, int w) {
b54          rootify(w), access(v);
264          t[v].ch[0] = t[t[v].ch[0]].p = -1;
cbb      }
031      void cut(int v, int w) {
b0f          int id = aresta[make_pair(v, w)];
a4a          cut_(v, id), cut_(id, w);
cbb      }
bbb      int lca(int v, int w) {
5e3          access(v);
a8b          return access(w);
cbb      }
cbb      }

```

## 2.29 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

1ef namespace lct {
3c9     struct node {
19f         int p, ch[2];
810         ll val, sub;
aa6         bool rev;
e4d         int sz;
4e4         ll lazy;
f93         node() {}
aa0         node(int v) : p(-1), val(v), sub(v), rev(0),
sz(1), lazy(0) {
b07             ch[0] = ch[1] = -1;
cbb         }
214     };

5f3     node t[MAX];

95a     void prop(int x) {
dc1         if (t[x].lazy) {
9f7             t[x].val += t[x].lazy, t[x].sub +=
t[x].lazy*t[x].sz;
edc             if (t[x].ch[0]+1) t[t[x].ch[0]].lazy +=
t[x].lazy;
942             if (t[x].ch[1]+1) t[t[x].ch[1]].lazy +=
t[x].lazy;
cbb         }
aa2         if (t[x].rev) {
f95             swap(t[x].ch[0], t[x].ch[1]);
379             if (t[x].ch[0]+1) t[t[x].ch[0]].rev ^= 1;
c3d             if (t[x].ch[1]+1) t[t[x].ch[1]].rev ^= 1;
cbb         }
```

```
230         t[x].lazy = 0, t[x].rev = 0;
cbb     }
564     void update(int x) {
ec2         t[x].sz = 1, t[x].sub = t[x].val;
8ca         for (int i = 0; i < 2; i++) if (t[x].ch[i]+1) {
621             prop(t[x].ch[i]);
c4f             t[x].sz += t[t[x].ch[i]].sz;
269             t[x].sub += t[t[x].ch[i]].sub;
cbb         }
cbb     }
971     bool is_root(int x) {
657         return t[x].p == -1 or (t[t[x].p].ch[0] != x and
t[t[x].p].ch[1] != x);
cbb     }
ed6     void rotate(int x) {
497         int p = t[x].p, pp = t[p].p;
fc4         if (!is_root(p)) t[pp].ch[t[pp].ch[1] == p] = x;
251         bool d = t[p].ch[0] == x;
461         t[p].ch[!d] = t[x].ch[d], t[x].ch[d] = p;
a76         if (t[p].ch[!d]+1) t[t[p].ch[!d]].p = p;
8fa         t[x].p = pp, t[p].p = x;
444         update(p), update(x);
cbb     }
238     int splay(int x) {
18c         while (!is_root(x)) {
497             int p = t[x].p, pp = t[p].p;
77b             if (!is_root(p)) prop(pp);
be5             prop(p), prop(x);
0c5             if (!is_root(p)) rotate((t[pp].ch[0] ==
p)^(t[p].ch[0] == x) ? x : p);
64f             rotate(x);
cbb         }
aab         return prop(x), x;
cbb     }
f16     int access(int v) {
0eb         int last = -1;
d9f         for (int w = v; w+1; update(last = w), splay(v),
w = t[v].p)
024             splay(w), t[w].ch[1] = (last == -1 ? -1 : v);
3d3         return last;
cbb     }
f17     void make_tree(int v, int w) { t[v] = node(w); }
```

```

e89     int find_root(int v) {
13f         access(v), prop(v);
9f0         while (t[v].ch[0]+1) v = t[v].ch[0], prop(v);
637     return splay(v);
cbb     }
f94     bool connected(int v, int w) {
2cf         access(v), access(w);
b9b         return v == w ? true : t[v].p != -1;
cbb     }
277     void rootify(int v) {
5e3         access(v);
a02         t[v].rev ^= 1;
cbb     }
971     ll query(int v, int w) {
b54         rootify(w), access(v);
249         return t[v].sub;
cbb     }
3fa     void update(int v, int w, int x) {
b54         rootify(w), access(v);
12c         t[v].lazy += x;
cbb     }
142     void link(int v, int w) {
821         rootify(w);
389         t[w].p = v;
cbb     }
031     void cut(int v, int w) {
b54         rootify(w), access(v);
264         t[v].ch[0] = t[t[v].ch[0]].p = -1;
cbb     }
bbb     int lca(int v, int w) {
5e3         access(v);
a8b         return access(w);
cbb     }
cbb }

```

## 2.30 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

919 struct lb_max_flow : dinic {
5ce     vector<int> d;
331     lb_max_flow(int n) : dinic(n + 2), d(n, 0) {}
b12     void add(int a, int b, int l, int r) {
c97         d[a] -= l;
f1b         d[b] += l;
017         dinic::add(a, b, r - l);
cbb     }
087     void add(int a, int b, int c) {
107         dinic::add(a, b, c);
cbb     }
7a1     bool has_circulation() {
50c         int n = d.size();

854         ll cost = 0;
603         for (int i = 0; i < n; i++) {
c69             if (d[i] > 0) {
f56                 cost += d[i];
d06                 dinic::add(n, i, d[i]);
9c7             } else if (d[i] < 0) {
76b                 dinic::add(i, n+1, -d[i]);
cbb             }
cbb         }

283         return (dinic::max_flow(n, n+1) == cost);
cbb     }
7bd     bool has_flow(int src, int snk) {
65d         dinic::add(snk, src, INF);
e40         return has_circulation();
cbb     }
4eb     ll max_flow(int src, int snk) {
ee8         if (!has_flow(src, snk)) return -1;
ea5         dinic::F = 0;
626         return dinic::max_flow(src, snk);
cbb     }
214 };

```



## 2.31 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

123 template<typename T> struct mcmf {
670     struct edge {
b75         int to, rev, flow, cap; // para, id da reversa,
        fluxo, capacidade
7f9         bool res; // se eh reversa
635         T cost; // custo da unidade de fluxo
892         edge() : to(0), rev(0), flow(0), cap(0),
        cost(0), res(false) {}
1d7         edge(int to_, int rev_, int flow_, int cap_, T
        cost_, bool res_)
f8d             : to(to_), rev(rev_), flow(flow_),
        cap(cap_), res(res_), cost(cost_) {}
214     };

002     vector<vector<edge>> g;
168     vector<int> par_idx, par;
f1e     T inf;
a03     vector<T> dist;

b22     mcmf(int n) : g(n), par_idx(n), par(n),
        inf(numeric_limits<T>::max()/3) {}

91c     void add(int u, int v, int w, T cost) { // de u pra
        v com cap w e custo cost
2fc         edge a = edge(v, g[v].size(), 0, w, cost, false);
234         edge b = edge(u, g[u].size(), 0, 0, -cost, true);

b24         g[u].push_back(a);
```

```
c12         g[v].push_back(b);
cbb     }

8bc     vector<T> spfa(int s) { // nao precisa se nao tiver
        custo negativo
871         deque<int> q;
3d1         vector<bool> is_inside(g.size(), 0);
577         dist = vector<T>(g.size(), inf);

a93         dist[s] = 0;
a30         q.push_back(s);
ecb         is_inside[s] = true;

14d         while (!q.empty()) {
b1e             int v = q.front();
ced             q.pop_front();
48d             is_inside[v] = false;

76e             for (int i = 0; i < g[v].size(); i++) {
9d4                 auto [to, rev, flow, cap, res, cost] =
        g[v][i];
e61                 if (flow < cap and dist[v] + cost <
        dist[to]) {
943                     dist[to] = dist[v] + cost;

ed6                     if (is_inside[to]) continue;
020                     if (!q.empty() and dist[to] >
        dist[q.front()]) q.push_back(to);
b33                     else q.push_front(to);
b52                     is_inside[to] = true;
cbb                 }
cbb             }
cbb         }
8d7         return dist;
cbb     }

2a2     bool dijkstra(int s, int t, vector<T>& pot) {
489         priority_queue<pair<T, int>, vector<pair<T,
        int>>, greater<>> q;
577         dist = vector<T>(g.size(), inf);
a93         dist[s] = 0;
115         q.emplace(0, s);
402         while (q.size()) {
```

```

91b         auto [d, v] = q.top();
833         q.pop();
68b         if (dist[v] < d) continue;
76e         for (int i = 0; i < g[v].size(); i++) {
9d4             auto [to, rev, flow, cap, res, cost] =
                g[v][i];
e8c             cost += pot[v] - pot[to];
e61             if (flow < cap and dist[v] + cost <
                dist[to]) {
943                 dist[to] = dist[v] + cost;
441                 q.emplace(dist[to], to);
88b                 par_idx[to] = i, par[to] = v;
cbb             }
cbb         }
cbb     }
1d4     return dist[t] < inf;
cbb }

3d2     pair<int, T> min_cost_flow(int s, int t, int flow =
INF) {
3dd         vector<T> pot(g.size(), 0);
9e4         pot = spfa(s); // mudar algoritmo de caminho
                minimo aqui

d22         int f = 0;
ce8         T ret = 0;
4a0         while (f < flow and dijkstra(s, t, pot)) {
bda             for (int i = 0; i < g.size(); i++)
d2a                 if (dist[i] < inf) pot[i] += dist[i];

71b             int mn_flow = flow - f, u = t;
045             while (u != s){
90f                 mn_flow = min(mn_flow,
07d                     g[par[u]][par_idx[u]].cap -
                g[par[u]][par_idx[u]].flow);
3d1                 u = par[u];
cbb             }

1f2             ret += pot[t] * mn_flow;

476             u = t;
045             while (u != s) {

```

```

e09                 g[par[u]][par_idx[u]].flow += mn_flow;
d98                 g[u][g[par[u]][par_idx[u]].rev].flow -=
                mn_flow;
3d1                 u = par[u];
cbb             }

04d             f += mn_flow;
cbb         }

15b         return make_pair(f, ret);
cbb     }

        // Opcional: retorna as arestas originais por onde
        passa flow = cap
182     vector<pair<int,int>> recover() {
24a         vector<pair<int,int>> used;
2a4         for (int i = 0; i < g.size(); i++) for (edge e :
                g[i])
587             if(e.flow == e.cap && !e.res)
                used.push_back({i, e.to});
f6b         return used;
cbb     }
214 };

```

## 2.32 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
47d vector<int> to_prufer(vector<pair<int, int>> tree) {
1fa     int n = tree.size()+1;
2cf     vector<int> d(n, 0);
4aa     vector<vector<int>> g(n);
f87     for (auto [a, b] : tree) d[a]++, d[b]++,
f60         g[a].push_back(b), g[b].push_back(a);
c5a     vector<int> pai(n, -1);

```

```

260     queue<int> q; q.push(n-1);
402     while (q.size()) {
be1         int u = q.front(); q.pop();
34c         for (int v : g[u]) if (v != pai[u])
9c9             pai[v] = u, q.push(v);
cbb     }
399     int idx, x;
897     idx = x = find(d.begin(), d.end(), 1) - d.begin();
4b8     vector<int> ret;
b28     for (int i = 0; i < n-2; i++) {
d4b         int y = pai[x];
e81         ret.push_back(y);
666         if (--d[y] == 1 and y < idx) x = y;
367         else idx = x = find(d.begin()+idx+1, d.end(), 1)
- d.begin();
cbb     }
edf     return ret;
cbb }

// 765413
4d8 vector<pair<int, int>> from_prufer(vector<int> p) {
455     int n = p.size()+2;
126     vector<int> d(n, 1);
650     for (int i : p) d[i]++;
85b     p.push_back(n-1);
399     int idx, x;
897     idx = x = find(d.begin(), d.end(), 1) - d.begin();
1df     vector<pair<int, int>> ret;
b06     for (int y : p) {
dab         ret.push_back({x, y});
666         if (--d[y] == 1 and y < idx) x = y;
367         else idx = x = find(d.begin()+idx+1, d.end(), 1)
- d.begin();
cbb     }
edf     return ret;
cbb }

```

## 2.33 Sack (DSU em arvores)

```

// Responde queries de todas as sub-arvores
// offline
//

```

```

// 0(n log(n))
// bb361f

6bf int sz[MAX], cor[MAX], cnt[MAX];
042 vector<int> g[MAX];

6df void build(int k, int d=0) {
e8f     sz[k] = 1;
01a     for (auto& i : g[k]) {
30f         build(i, d+1); sz[k] += sz[i];
925         if (sz[i] > sz[g[k][0]]) swap(i, g[k][0]);
cbb     }
cbb }

74f void compute(int k, int x, bool dont=1) {
de9     cnt[cor[k]] += x;
828     for (int i = dont; i < g[k].size(); i++)
b5c         compute(g[k][i], x, 0);
cbb }

dc4 void solve(int k, bool keep=0) {
32a     for (int i = int(g[k].size())-1; i >= 0; i--)
b4c         solve(g[k][i], !i);
4a0     compute(k, 1);

// agora cnt[i] tem quantas vezes a cor
// i aparece na sub-arvore do k

830     if (!keep) compute(k, -1, 0);
cbb }

```

## 2.34 Tarjan para SCC

```

// 0(n + m)
// 573bfa

042 vector<int> g[MAX];
4ce stack<int> s;
a42 int vis[MAX], comp[MAX];
3fd 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
f32 int dfs(int i, int& t) {
cf0     int lo = id[i] = t++;
18e     s.push(i);
0c2     vis[i] = 2;

48e     for (int j : g[i]) {
740         if (!vis[j]) lo = min(lo, dfs(j, t));
994         else if (vis[j] == 2) lo = min(lo, id[j]);
cbb     }

    // aresta de i pro pai eh uma ponte (no caso nao
    direcionado)
3de     if (lo == id[i]) while (1) {
3c3         int u = s.top(); s.pop();
9c5         vis[u] = 1, comp[u] = i;
2ef         if (u == i) break;
cbb     }

253     return lo;
cbb }

f93 void tarjan(int n) {
6bb     int t = 0;
991     for (int i = 0; i < n; i++) vis[i] = 0;

3be     for (int i = 0; i < n; i++) if (!vis[i]) dfs(i, t);
cbb }

```

## 2.35 Topological Sort

```

// Retorna uma ordenacao topologica de g
// Se g nao for DAG retorna um vetor vazio
//
// O(n + m)
// bdc95e

```

```

042 vector<int> g[MAX];

b6a vector<int> topo_sort(int n) {

```

```

46e     vector<int> ret(n,-1), vis(n,0);

f51     int pos = n-1, dag = 1;
36d     function<void(int)> dfs = [&](int v) {
cca         vis[v] = 1;
440         for (auto u : g[v]) {
152             if (vis[u] == 1) dag = 0;
532             else if (!vis[u]) dfs(u);
cbb         }
d44         ret[pos--] = v, vis[v] = 2;
214     };

158     for (int i = 0; i < n; i++) if (!vis[i]) dfs(i);

d8f     if (!dag) ret.clear();
edf     return ret;
cbb }

```

## 2.36 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

```

```

76a namespace cover {
5a4     const int MAX = 96;
042     vector<int> g[MAX];
823     bitset<MAX> bs[MAX];
1a8     int n;

697     void add(int i, int j) {
bd0         if (i == j) return;
78c         n = max({n, i+1, j+1});
200         bs[i][j] = bs[j][i] = 1;
cbb     }

```

```

6c0     int rec(bitset<MAX> m) {
1a4         int ans = 0;
25b         for (int x = 0; x < n; x++) if (m[x]) {

```

```

002         bitset<MAX> comp;
4bf         function<void(int)> dfs = [&](int i) {
b96             comp[i] = 1, m[i] = 0;
0c3             for (int j : g[i]) if (m[j]) dfs(j);
214         };
963         dfs(x);

d34         int ma, deg = -1, cyc = 1;
417         for (int i = 0; i < n; i++) if (comp[i]) {
d0b             int d = (bs[i]&comp).count();
18a             if (d <= 1) cyc = 0;
c1f             if (d > deg) deg = d, ma = i;
cbb         }
269         if (deg <= 2) { // caminho ou ciclo
340             ans += (comp.count() + cyc) / 2;
5e2             continue;
cbb         }
3f9         comp[ma] = 0;

// ou ta no cover, ou nao ta no cover
1dd         ans += min(1 + rec(comp), deg + rec(comp & ~
        bs[ma]));
cbb     }
ba7     return ans;
cbb }

f5c     int solve() {
3c5         bitset<MAX> m;
603         for (int i = 0; i < n; i++) {
939             m[i] = 1;
f90             for (int j = 0; j < n; j++)
741                 if (bs[i][j]) g[i].push_back(j);
cbb         }
4f9         return rec(m);
cbb     }
cbb }

```

## 2.37 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 de virt 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

b36     vector<pair<int, int>> virt[MAX];

d41     #warning lembrar de buildar o LCA antes
c14     int build_virt(vector<int> v) {
b46         auto cmp = [&](int i, int j) { return lca::pos[i] <
            lca::pos[j]; };
074         sort(v.begin(), v.end(), cmp);
e85         for (int i = v.size()-1; i; i--)
            v.push_back(lca::lca(v[i], v[i-1]));
074         sort(v.begin(), v.end(), cmp);
d76         v.erase(unique(v.begin(), v.end()), v.end());
37c         for (int i = 0; i < v.size(); i++)
            virt[v[i]].clear();
197         for (int i = 1; i < v.size(); i++)
            virt[lca::lca(v[i-1], v[i])].clear();
ad7         for (int i = 1; i < v.size(); i++) {
51b             int parent = lca::lca(v[i-1], v[i]);
290             int d = lca::dist(parent, v[i]);
d41             #warning soh to colocando aresta descendo
4d0             virt[parent].emplace_back(v[i], d);
cbb         }
832         return v[0];
cbb     }

```

## 3 Problemas

### 3.1 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

a6a template<typename T> struct hungarian {
1a8     int n;
a08     vector<vector<T>> a;
f36     vector<T> u, v;
5ff     vector<int> p, way;
f1e     T inf;

c3f     hungarian(int n_) : n(n_), u(n+1), v(n+1), p(n+1),
way(n+1) {
b2f         a = vector<vector<T>>(n, vector<T>(n));
1f3         inf = numeric_limits<T>::max();
cbb     }
d67     pair<T, vector<int>> assignment() {
78a         for (int i = 1; i <= n; i++) {
8c9             p[0] = i;
625             int j0 = 0;
ce7             vector<T> minv(n+1, inf);
241             vector<int> used(n+1, 0);
016             do {
472                 used[j0] = true;
d24                 int i0 = p[j0], j1 = -1;
7e5                 T delta = inf;
9ac                 for (int j = 1; j <= n; j++) if
(!used[j]) {
7bf                     T cur = a[i0-1][j-1] - u[i0] - v[j];
9f2                     if (cur < minv[j]) minv[j] = cur,
way[j] = j0;
821                     if (minv[j] < delta) delta =
minv[j], j1 = j;
cbb                 }
f63                 for (int j = 0; j <= n; j++)
2c5                     if (used[j]) u[p[j]] += delta, v[j]
-= delta;
6ec                     else minv[j] -= delta;

```

```

6d4             j0 = j1;
233         } while (p[j0] != 0);
016         do {
4c5             int j1 = way[j0];
0d7             p[j0] = p[j1];
6d4             j0 = j1;
ca1         } while (j0);
cbb     }
306     vector<int> ans(n);
6db     for (int j = 1; j <= n; j++) ans[p[j]-1] = j-1;
da3     return make_pair(-v[0], ans);
cbb     }
214 };

```

### 3.2 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

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

042 vector<int> g[MAX];

c54 namespace LCA { ... }

249 int in[MAX], out[MAX], vtx[2 * MAX];
81b bool on[MAX];

4c3 int dif, freq[MAX];
9e2 vector<int> w;

d9a void dfs(int v, int p, int &t) {
659     vtx[t] = v, in[v] = t++;
18e     for (int u : g[v]) if (u != p) {
c53         dfs(u, v, t);
cbb     }
217     vtx[t] = v, out[v] = t++;
cbb }

```

```

e5f void update(int p) { // faca alteracoes aqui
bbc     int v = vtx[p];
0ec     if (not on[v]) { // insere vtx v
31c         dif += (freq[w[v]] == 0);
b20         freq[w[v]]++;
cbb     }
4e6     else { // retira o vertice v
0a9         dif -= (freq[w[v]] == 1);
fd3         freq[w[v]]--;
cbb     }
73e     on[v] = not on[v];
cbb }

a3a vector<tuple<int, int, int>> build_queries(const
vector<pair<int, int>>& q) {
ea6     LCA::build(0);
f77     vector<tuple<int, int, int>> ret;
aa9     for (auto [l, r] : q){
d24         if (in[r] < in[l]) swap(l, r);
6f9         int p = LCA::lca(l, r);
826         int init = (p == l) ? in[l] : out[l];
07a         ret.emplace_back(init, in[r], in[p]);
cbb     }
edf     return ret;
cbb }

f31 vector<int> mo_tree(const vector<pair<int, int>>& vq){
6bb     int t = 0;
dab     dfs(0, -1, t);

af1     auto q = build_queries(vq);

f48     vector<int> ord(q.size());
be8     iota(ord.begin(), ord.end(), 0);
d01     sort(ord.begin(), ord.end(), [&] (int l, int r) {
d8d         int bl = get<0>(q[l]) / SQ, br = get<0>(q[r]) /
SQ;
596         if (bl != br) return bl < br;
158         else if (bl % 2 == 1) return get<1>(q[l]) <
get<1>(q[r]);
f1d         else return get<1>(q[l]) > get<1>(q[r]);
c0c     });

```

```

80e     memset(freq, 0, sizeof freq);
bf6     dif = 0;

ff2     vector<int> ret(q.size());
3d9     int l = 0, r = -1;
8b0     for (int i : ord) {
3c7         auto [ql, qr, qp] = q[i];
af7         while (r < qr) update(++r);
d6b         while (l > ql) update(--l);
951         while (l < ql) update(l++);
6a1         while (r > qr) update(r--);

3d8         if (qp < l or qp > r) { // se LCA estah entre as
pontas
74b             update(qp);
2e1             ret[i] = dif;
74b             update(qp);
cbb         }
0fe         else ret[i] = dif;
cbb     }
edf     return ret;
cbb }

```

### 3.3 Angle Range Intersection

```

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

32a struct angle_range {
75e     static constexpr ld ALL = 1e9, NIL = -1e9;
395     ld l, r;
c77     angle_range() : l(ALL), r(ALL) {}
894     angle_range(ld l_, ld r_) : l(l_), r(r_) { fix(l),
fix(r); }

4ee     void fix(ld& theta) {
da7         if (theta == ALL or theta == NIL) return;

```

```

323         if (theta > 2*pi) theta -= 2*pi;
868         if (theta < 0) theta += 2*pi;
cbb     }
2ee     bool empty() { return l == NIL; }
931     bool contains(ld q) {
40f         fix(q);
4d7         if (l == ALL) return true;
fec         if (l == NIL) return false;
6a6         if (l < r) return l < q and q < r;
075         return q > l or q < r;
cbb     }
9c7     friend angle_range operator &(angle_range p,
angle_range q) {
743         if (p.l == ALL or q.l == NIL) return q;
20f         if (q.l == ALL or p.l == NIL) return p;
7d5         if (p.l > p.r and q.l > q.r) return {max(p.l,
q.l) , min(p.r, q.r)};
aa6         if (q.l > q.r) swap(p.l, q.l), swap(p.r, q.r);
8d8         if (p.l > p.r) {
249             if (q.r > p.l) return {max(q.l, p.l) , q.r};
6f7             else if (q.l < p.r) return {q.l, min(q.r,
p.r)};
270             return {NIL, NIL};
cbb         }
5a8         if (max(p.l, q.l) > min(p.r, q.r)) return {NIL,
NIL};
bcb         return {max(p.l, q.l), min(p.r, q.r)};
cbb     }
214 };

```

### 3.4 Area da Uniao de Retangulos

```

// O(n log(n))
// bea565

aa4 namespace seg {
6b3     pair<int, ll> seg[4*MAX];
b1b     ll lazy[4*MAX], *v;
1a8     int n;

e01     pair<int, ll> merge(pair<int, ll> l, pair<int, ll>
r){

```

```

719         if (l.second == r.second) return
{1.first+r.first, l.second};
53b         else if (l.second < r.second) return l;
aa0         else return r;
cbb     }

6fc     pair<int, ll> build(int p=1, int l=0, int r=n-1) {
3c7         lazy[p] = 0;
bf8         if (l == r) return seg[p] = {1, v[l]};
ee4         int m = (l+r)/2;
432         return seg[p] = merge(build(2*p, l, m),
build(2*p+1, m+1, r));
cbb     }
d9e     void build(int n2, ll* v2) {
680         n = n2, v = v2;
6f2         build();
cbb     }
ceb     void prop(int p, int l, int r) {
208         seg[p].second += lazy[p];
2c9         if (l != r) lazy[2*p] += lazy[p], lazy[2*p+1] +=
lazy[p];
3c7         lazy[p] = 0;
cbb     }
693     pair<int, ll> query(int a, int b, int p=1, int l=0,
int r=n-1) {
6b9         prop(p, l, r);
527         if (a <= l and r <= b) return seg[p];
9b7         if (b < l or r < a) return {0, LINF};
ee4         int m = (l+r)/2;
eeb         return merge(query(a, b, 2*p, l, m), query(a, b,
2*p+1, m+1, r));
cbb     }
07c     pair<int, ll> update(int a, int b, int x, int p=1,
int l=0, int r=n-1) {
6b9         prop(p, l, r);
9a3         if (a <= l and r <= b) {
b94             lazy[p] += x;
6b9             prop(p, l, r);
534             return seg[p];
cbb         }
e9f         if (b < l or r < a) return seg[p];
ee4         int m = (l+r)/2;

```



```

086         return seg[p] = merge(update(a, b, x, 2*p, l, m),
579         update(a, b, x, 2*p+1, m+1, r));
cbb     }
214 };

eb5 ll seg_vec[MAX];

8be ll area_sq(vector<pair<pair<int, int>, pair<int, int>>>
&sq){
28c     vector<pair<pair<int, int>, pair<int, int>>> up;
60a     for (auto it : sq){
619         int x1, y1, x2, y2;
ae0         tie(x1, y1) = it.first;
68e         tie(x2, y2) = it.second;
80f         up.push_back({x1+1, 1}, {y1, y2});
aee         up.push_back({x2+1, -1}, {y1, y2});
cbb     }
092     sort(up.begin(), up.end());
049     memset(seg_vec, 0, sizeof seg_vec);
6fe     ll H_MAX = MAX;
156     seg::build(H_MAX-1, seg_vec);
7ba     auto it = up.begin();
04b     ll ans = 0;
f14     while (it != up.end()){
07f         ll L = (*it).first.first;
718         while (it != up.end() && (*it).first.first == L){
127             int x, inc, y1, y2;
d35             tie(x, inc) = it->first;
d3d             tie(y1, y2) = it->second;
5d1             seg::update(y1+1, y2, inc);
40d             it++;
cbb         }
852         if (it == up.end()) break;
d8a         ll R = (*it).first.first;

f59         ll W = R-L;
efd         auto jt = seg::query(0, H_MAX-1);
91a         ll H = H_MAX - 1;
e8a         if (jt.second == 0) H -= jt.first;
8df         ans += W*H;
cbb     }
ba7     return ans;

```

```
cbb }
```

### 3.5 Area Maxima de Histograma

```

// Assume que todas as barras tem largura 1,
// e altura dada no vetor v
//
// 0(n)
// e43846

15e ll area(vector<int> v) {
b73     ll ret = 0;
4ce     stack<int> s;
        // valores iniciais pra dar tudo certo
447     v.insert(v.begin(), -1);
d56     v.insert(v.end(), -1);
1f8     s.push(0);

0be     for(int i = 0; i < (int) v.size(); i++) {
78e         while (v[s.top()] > v[i]) {
265             ll h = v[s.top()]; s.pop();
de1             ret = max(ret, h * (i - s.top() - 1));
cbb         }
18e         s.push(i);
cbb     }

edf     return ret;
cbb }

```

### 3.6 Binomial modular

```

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

97c ll divi[MAX];

398 ll expo(ll a, ll b, ll m) {
1c1     if (!b) return 1;
399     ll ans = expo(a*a%m, b/2, m);
751     if (b%2) ans *= a;

```

```

2e9     return ans%m;
cbb }

f0a ll inv(ll a, ll b){
bca     return 1<a ? b - inv(b%a,a)*b/a : 1;
cbb }

153 template<typename T> tuple<T, T, T> ext_gcd(T a, T b) {
3bd     if (!a) return {b, 0, 1};
550     auto [g, x, y] = ext_gcd(b%a, a);
c59     return {g, y - b/a*x, x};
cbb }

bfe template<typename T = ll> struct crt {
627     T a, m;

5f3     crt() : a(0), m(1) {}
7eb     crt(T a_, T m_) : a(a_), m(m_) {}
911     crt operator * (crt C) {
238         auto [g, x, y] = ext_gcd(m, C.m);
dc0         if ((a - C.a) % g) a = -1;
4f9         if (a == -1 or C.a == -1) return crt(-1, 0);
d09         T lcm = m/g*C.m;
eb2         T ans = a + (x*(C.a-a)/g % (C.m/g))*m;
d8d         return crt((ans % lcm + lcm) % lcm, lcm);
cbb     }
214 };

6f2 pair<ll, ll> divide_show(ll n, int p, int k, int pak) {
4f7     if (n == 0) return {0, 1};
d02     ll blocos = n/pak, falta = n%pak;
2ce     ll periodo = divi[pak], resto = divi[falta];
616     ll r = expo(periodo, blocos, pak)*resto%pak;

445     auto rec = divide_show(n/p, p, k, pak);
a51     ll y = n/p + rec.first;
bb9     r = r*rec.second % pak;

90f     return {y, r};
cbb }

6e6 ll solve_pak(ll n, ll x, int p, int k, int pak) {

```

```

d34     divi[0] = 1;
f2b     for (int i = 1; i <= pak; i++) {
901         divi[i] = divi[i-1];
840         if (i%p) divi[i] = divi[i] * i % pak;
cbb     }

4ac     auto dn = divide_show(n, p, k, pak), dx =
        divide_show(x, p, k, pak),
162         dnx = divide_show(n-x, p, k, pak);
768     ll y = dn.first-dx.first-dnx.first, r =
b64         (dn.second*inv(dx.second,
        pak)%pak)*inv(dnx.second, pak)%pak;
035     return expo(p, y, pak) * r % pak;
cbb }

9dd ll solve(ll n, ll x, int mod) {
490     vector<pair<int, int>> f;
c3b     int mod2 = mod;
7b4     for (int i = 2; i*i <= mod2; i++) if (mod2%i==0) {
aff         int c = 0;
75b         while (mod2%i==0) mod2 /= i, c++;
2a1         f.push_back({i, c});
cbb     }
0ff     if (mod2 > 1) f.push_back({mod2, 1});
e96     crt ans(0, 1);
a13     for (int i = 0; i < f.size(); i++) {
702         int pak = 1;
7e4         for (int j = 0; j < f[i].second; j++) pak *=
            f[i].first;
304         ans = ans * crt(solve_pak(n, x, f[i].first,
            f[i].second, pak), pak);
cbb     }
5fb     return ans.a;
cbb }

```

### 3.7 Closest pair of points

```

// O(nlogn)
// f90265

915 pair<pt, pt> closest_pair_of_points(vector<pt> v) {
3d2     int n = v.size();

```

```

fca      sort(v.begin(), v.end());
31c      for (int i = 1; i < n; i++) if (v[i] == v[i-1])
          return {v[i-1], v[i]};
c20      auto cmp_y = [&](const pt &l, const pt &r) {
b53          if (l.y != r.y) return l.y < r.y;
920          return l.x < r.x;
214      };
62e      set<pt, decltype(cmp_y)> s(cmp_y);
3d9      int l = 0, r = -1;
6a2      ll d2_min = numeric_limits<ll>::max();
4d5      pt pl, pr;
bd1      const int magic = 5;
a55      while (r+1 < n) {
7f1          auto it = s.insert(v[++r]).first;
c92          int cnt = magic/2;
773          while (cnt-- and it != s.begin()) it--;
a01          cnt = 0;
d68          while (cnt++ < magic and it != s.end()) {
f19              if (!((*it) == v[r])) {
67e                  ll d2 = dist2(*it, v[r]);
74e                  if (d2_min > d2) {
229                      d2_min = d2;
841                      pl = *it;
4f2                      pr = v[r];
cbb              }
cbb          }
40d          it++;
cbb      }
eb0      while (l < r and sq(v[l].x-v[r].x) > d2_min)
          s.erase(v[l++]);
cbb      }
c74      return {pl, pr};
cbb  }

```

### 3.8 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
```

```

615 vector<int> coloring(vector<pair<int, int>>& v) {
3d2     int n = v.size();
c08     vector<pair<int, pair<int, int>>> ev;
603     for (int i = 0; i < n; i++) {
150         ev.push_back({v[i].first, {1, i}});
cda         ev.push_back({v[i].second, {0, i}});
cbb     }
49e     sort(ev.begin(), ev.end());
360     vector<int> ans(n), avl(n);
265     for (int i = 0; i < n; i++) avl.push_back(n-i);
4bf     for (auto i : ev) {
cbe         if (i.second.first == 1) {
021             ans[i.second.second] = avl.back();
a00             avl.pop_back();
296         } else avl.push_back(ans[i.second.second]);
cbb     }
ba7     return ans;
cbb }

```

### 3.9 Conectividade Dinamica

```

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

```

```

8f2 typedef pair<int, int> T;

1cd namespace data {
553     int n, ans;
573     int p[MAX], sz[MAX];
ee6     stack<int> S;

e5b     void build(int n2) {
1e3         n = n2;
8a6         for (int i = 0; i < n; i++) p[i] = i, sz[i] = 1;
0b2         ans = n;
cbb     }
1b1     int find(int k) {
006         while (p[k] != k) k = p[k];

```

```

839         return k;
cbb     }
072 void add(T x) {
700     int a = x.first, b = x.second;
605     a = find(a), b = find(b);
843     if (a == b) return S.push(-1);
e7d     ans--;
3c6     if (sz[a] > sz[b]) swap(a, b);
4c2     S.push(a);
582     sz[b] += sz[a];
84b     p[a] = b;
cbb     }
5eb int query() {
ba7     return ans;
cbb     }
5cf void rollback() {
465     int u = S.top(); S.pop();
61c     if (u == -1) return;
270     sz[p[u]] -= sz[u];
546     p[u] = u;
0df     ans++;
cbb     }
214 };

357 int ponta[MAX]; // outra ponta do intervalo ou -1 se for
    query
4f0 int ans[MAX], n, q;
487 T qu[MAX];

47b void solve(int l = 0, int r = q-1) {
0b1     if (l >= r) {
8c0         ans[l] = data::query(); // agora a estrutura ta
        certa
505         return;
cbb     }
962     int m = (l+r)/2, qnt = 1;
fc7     for (int i = m+1; i <= r; i++) if (ponta[i]+1 and
        ponta[i] < l)
37d         data::add(qu[i]), qnt++;
221     solve(l, m);
593     while (--qnt) data::rollback();
a2c     for (int i = l; i <= m; i++) if (ponta[i]+1 and

```

```

        ponta[i] > r)
37d         data::add(qu[i]), qnt++;
37b     solve(m+1, r);
281     while (qnt--> 0) data::rollback();
cbb }

```

### 3.10 Conectividade Dinamica 2

```

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

1ef namespace lct {
3c9     struct node {
19f         int p, ch[2];
a2a         int val, sub;
aa6         bool rev;
f93         node() {}
54e         node(int v) : p(-1), val(v), sub(v), rev(0) {
            ch[0] = ch[1] = -1; }
214     };

c53     node t[2*MAX]; // MAXN + MAXQ
99e     map<pair<int, int>, int> aresta;
e4d     int sz;

95a     void prop(int x) {
aa2         if (t[x].rev) {
f95             swap(t[x].ch[0], t[x].ch[1]);
379             if (t[x].ch[0]+1) t[t[x].ch[0]].rev ^= 1;
c3d             if (t[x].ch[1]+1) t[t[x].ch[1]].rev ^= 1;
cbb         }
693         t[x].rev = 0;
cbb     }
564     void update(int x) {
e8d         t[x].sub = t[x].val;
8ca         for (int i = 0; i < 2; i++) if (t[x].ch[i]+1) {
621             prop(t[x].ch[i]);
78d             t[x].sub = min(t[x].sub, t[t[x].ch[i]].sub);
cbb         }
cbb     }
571     bool is_root(int x) {

```

```

657     return t[x].p == -1 or (t[t[x].p].ch[0] != x and
t[t[x].p].ch[1] != x);
cbb }
ed6 void rotate(int x) {
497     int p = t[x].p, pp = t[p].p;
fc4     if (!is_root(p)) t[pp].ch[t[pp].ch[1] == p] = x;
251     bool d = t[p].ch[0] == x;
461     t[p].ch[!d] = t[x].ch[d], t[x].ch[d] = p;
a76     if (t[p].ch[!d]+1) t[t[p].ch[!d]].p = p;
8fa     t[x].p = pp, t[p].p = x;
444     update(p), update(x);
cbb }
238 int splay(int x) {
18c     while (!is_root(x)) {
497         int p = t[x].p, pp = t[p].p;
77b         if (!is_root(p)) prop(pp);
be5         prop(p), prop(x);
0c5         if (!is_root(p)) rotate((t[pp].ch[0] ==
p)^(t[p].ch[0] == x) ? x : p);
64f         rotate(x);
cbb     }
aab     return prop(x), x;
cbb }
f16 int access(int v) {
0eb     int last = -1;
d9f     for (int w = v; w+1; update(last = w), splay(v),
w = t[v].p)
024         splay(w), t[w].ch[1] = (last == -1 ? -1 : v);
3d3     return last;
cbb }
952 void make_tree(int v, int w=INF) { t[v] = node(w); }
82f bool conn(int v, int w) {
2cf     access(v), access(w);
b9b     return v == w ? true : t[v].p != -1;
cbb }
277 void rootify(int v) {
5e3     access(v);
a02     t[v].rev ^= 1;
cbb }
a1d int query(int v, int w) {
b54     rootify(w), access(v);
249     return t[v].sub;

```

```

cbb }
204 void link_(int v, int w) {
821     rootify(w);
389     t[w].p = v;
cbb }
6b8 void link(int v, int w, int x) { // v--w com peso x
379     int id = MAX + sz++;
110     aresta[make_pair(v, w)] = id;
ab6     make_tree(id, x);
c88     link_(v, id), link_(id, w);
cbb }
e63 void cut_(int v, int w) {
b54     rootify(w), access(v);
264     t[v].ch[0] = t[t[v].ch[0]].p = -1;
cbb }
031 void cut(int v, int w) {
b0f     int id = aresta[make_pair(v, w)];
a4a     cut_(v, id), cut_(id, w);
cbb }
cbb }

893 void dyn_conn() {
c5f     int n, q; cin >> n >> q;
d6e     vector<int> p(2*q, -1); // outra ponta do intervalo
b4f     for (int i = 0; i < n; i++) lct::make_tree(i);
fbf     vector<pair<int, int>> qu(q);
139     map<pair<int, int>, int> m;
abf     for (int i = 0; i < q; i++) {
3c2         char c; cin >> c;
ef6         if (c == '?') continue;
602         int a, b; cin >> a >> b; a--, b--;
d11         if (a > b) swap(a, b);
8a1         qu[i] = {a, b};
8d7         if (c == '+') {
94b             p[i] = i+q, p[i+q] = i;
906             m[make_pair(a, b)] = i;
9d9         } else {
412             int j = m[make_pair(a, b)];
ac2             p[i] = j, p[j] = i;
cbb         }
cbb     }
447     int ans = n;

```

```

abf     for (int i = 0; i < q; i++) {
87d         if (p[i] == -1) {
886             cout << ans << endl; // numero de comp
conexos
5e2             continue;
cbb         }
69d         int a = qu[i].first, b = qu[i].second;
c4d         if (p[i] > i) { // +
ac5             if (lct::conn(a, b)) {
18f                 int mi = lct::query(a, b);
993                 if (p[i] < mi) {
dd3                     p[p[i]] = p[i];
5e2                     continue;
cbb                 }
6f7                 lct::cut(qu[p[mi]].first,
qu[p[mi]].second), ans++;
6ea                 p[mi] = mi;
cbb             }
d1d             lct::link(a, b, p[i]), ans--;
cb5         } else if (p[i] != i) lct::cut(a, b), ans++; // -
cbb     }
cbb }

```

### 3.11 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

```

```

31e vector<int> ind_set(vector<tuple<int, int, int>>& v) {
b27     vector<tuple<int, int, int>> w;
f14     for (int i = 0; i < v.size(); i++) {
e85         w.push_back(tuple(get<0>(v[i]), 0, i));
6f0         w.push_back(tuple(get<1>(v[i]), 1, i));
cbb     }
d1d     sort(w.begin(), w.end());

```

```

844     vector<int> nxt(v.size());
c22     vector<pair<ll, int>> dp(v.size());
0eb     int last = -1;
723     for (auto [fim, t, i] : w) {
25a         if (t == 0) {
4ca             nxt[i] = last;
5e2             continue;
cbb         }
78b         dp[i] = {0, 0};
cb8         if (last != -1) dp[i] = max(dp[i], dp[last]);
911         pair<ll, int> pega = {get<2>(v[i]),
-(get<1>(v[i]) - get<0>(v[i]) + 1)};
5d3         if (nxt[i] != -1) pega.first +=
dp[nxt[i]].first, pega.second += dp[nxt[i]].second;
b08         if (pega > dp[i]) dp[i] = pega;
7cb         else nxt[i] = last;
381         last = i;
cbb     }
977     pair<ll, int> ans = {0, 0};
919     int idx = -1;
ceb     for (int i = 0; i < v.size(); i++) if (dp[i] > ans)
ans = dp[i], idx = i;
4b8     vector<int> ret;
fdd     while (idx != -1) {
d69         if (get<2>(v[idx]) > 0 and
a05             (nxt[idx] == -1 or get<1>(v[nxt[idx]]) <
get<0>(v[idx]))) ret.push_back(idx);
e4f         idx = nxt[idx];
cbb     }
0ea     sort(ret.begin(), ret.end());
edf     return ret;
cbb }

```

### 3.12 Distancia maxima entre dois pontos

```

// max_dist2(v) - 0(n log(n))
// max_dist_manhattan - 0(n)

// Quadrado da Distancia Euclidiana (precisa copiar
convex_hull, ccw e pt)
// bdace4
859 ll max_dist2(vector<pt> v) {

```

```

221     v = convex_hull(v);
a14     if (v.size() <= 2) return dist2(v[0], v[1%v.size()]);
04b     ll ans = 0;
323     int n = v.size(), j = 0;
603     for (int i = 0; i < n; i++) {
057         while (!ccw(v[(i+1)%n]-v[i], pt(0, 0),
v[(j+1)%n]-v[j])) j = (j+1)%n;
e7a         ans = max({ans, dist2(v[i], v[j]),
dist2(v[(i+1)%n], v[j])});
cbb     }
ba7     return ans;
cbb }

// Distancia de Manhattan
// 4e96f0
c51 template<typename T> T max_dist_manhattan(vector<pair<T,
T>> v) {
8eb     T min_sum, max_sum, min_dif, max_dif;
4f5     min_sum = max_sum = v[0].first + v[0].second;
271     min_dif = max_dif = v[0].first - v[0].second;
c25     for (auto [x, y] : v) {
1cb         min_sum = min(min_sum, x+y);
683         max_sum = max(max_sum, x+y);
782         min_dif = min(min_dif, x-y);
af7         max_dif = max(max_dif, x-y);
cbb     }
9f0     return max(max_sum - min_sum, max_dif - min_dif);
cbb }

```

### 3.13 Distinct Range Query

```

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

```

```

789 namespace perseg { };

53d int qt[MAX];

edc void build(vector<int>& v) {
3d2     int n = v.size();
16b     perseg::build(n);

```

```

663     map<int, int> last;
05e     int at = 0;
603     for (int i = 0; i < n; i++) {
817         if (last.count(v[i])) {
a58             perseg::update(last[v[i]], -1);
69a             at++;
cbb         }
4f2         perseg::update(i, 1);
460         qt[i] = ++at;
efe         last[v[i]] = i;
cbb     }
cbb }

9e3 int query(int l, int r) {
080     return perseg::query(l, r, qt[r]);
cbb }

```

### 3.14 Distinct Range Query com Update

```

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

```

```

774 #include <ext/pb_ds/assoc_container.hpp>
30f #include <ext/pb_ds/tree_policy.hpp>
0d7 using namespace __gnu_pbds;
4fc template <class T>
def     using ord_set = tree<T, null_type, less<T>,
rb_tree_tag,
3a1     tree_order_statistics_node_update>;

042 int v[MAX], n, nxt[MAX], prv[MAX];
f60 map<int, set<int>> > ocor;

e04 namespace bit {
686     ord_set<pair<int, int>> bit[MAX];

0a8     void build() {
3e1         for (int i = 1; i <= n; i++)
            bit[i].insert({nxt[i-1], i-1});
78a         for (int i = 1; i <= n; i++) {

```

```

edf         int j = i + (i&-i);
d03         if (j <= n) for (auto x : bit[i])
            bit[j].insert(x);
cbb     }
cbb     }
d3f     int pref(int p, int x) {
7c9         int ret = 0;
bbf         for (; p; p -= p&-p) ret +=
            bit[p].order_of_key({x, -INF});
edf         return ret;
cbb     }
d50     int query(int l, int r, int x) {
e55         return pref(r+1, x) - pref(l, x);
cbb     }
ff3     void update(int p, int x) {
f17         int p2 = p;
5ed         for (p++; p <= n; p += p&-p) {
ca8             bit[p].erase({nxt[p2], p2});
f6b             bit[p].insert({x, p2});
cbb         }
cbb     }
cbb }

0a8 void build() {
383     for (int i = 0; i < n; i++) nxt[i] = INF;
7b3     for (int i = 0; i < n; i++) prv[i] = -INF;
d07     vector<pair<int, int>> t;
348     for (int i = 0; i < n; i++) t.push_back({v[i], i});
3fd     sort(t.begin(), t.end());
603     for (int i = 0; i < n; i++) {
b40         if (i and t[i].first == t[i-1].first)
565             prv[t[i].second] = t[i-1].second;
a8b         if (i+1 < n and t[i].first == t[i+1].first)
12f             nxt[t[i].second] = t[i+1].second;
cbb     }

a23     for (int i = 0; i < n; i++) ocor[v[i]].insert(i);

1d7     bit::build();
cbb }

aae void muda(int p, int x) {

```

```

f92     bit::update(p, x);
c3d     nxt[p] = x;
cbb }

4ea int query(int a, int b) {
a0a     return b-a+1 - bit::query(a, b, b+1);
cbb }

ff3 void update(int p, int x) { // mudar valor na pos. p
    para x
c0b     if (prv[p] > -INF) muda(prv[p], nxt[p]);
4ae     if (nxt[p] < INF) prv[nxt[p]] = prv[p];

5bf     ocor[v[p]].erase(p);
4b4     if (!ocor[x].size()) {
19d         muda(p, INF);
8d4         prv[p] = -INF;
a69     } else if (*ocor[x].rbegin() < p) {
5b5         int i = *ocor[x].rbegin();
f64         prv[p] = i;
19d         muda(p, INF);
5f2         muda(i, p);
9d9     } else {
d46         int i = *ocor[x].lower_bound(p);
33f         if (prv[i] > -INF) {
f17             muda(prv[i], p);
8f9             prv[p] = prv[i];
94f         } else prv[p] = -INF;
523         prv[i] = p;
597         muda(p, i);
cbb     }
c96     v[p] = x; ocor[x].insert(p);
cbb }

```

### 3.15 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

e2a struct dominator_points {
baf     set<pair<int, int>> se;
4dd     multiset<int> quina;

a85     bool is_dominated(pair<int, int> p) {
80f         auto it = se.lower_bound(p);
633         if (it == se.end()) return 0;
ab4         return it->second >= p.second;
cbb     }
99b void mid(pair<int, int> a, pair<int, int> b, bool
rem) {
29a     pair<int, int> m = {a.first+1, b.second+1};
b19     int val = m.first + m.second;
638     if (!rem) quina.insert(val);
731     else quina.erase(quina.find(val));
cbb     }
7c4     bool insert(pair<int, int> p) {
fb4         if (is_dominated(p)) return 0;
80f         auto it = se.lower_bound(p);
ca9         if (it != se.begin() and it != se.end())
d4a             mid(*prev(it), *it, 1);
1fa         while (it != se.begin()) {
049             it--;
23c             if (it->second > p.second) break;
b86             if (it != se.begin()) mid(*prev(it), *it, 1);
316             it = se.erase(it);
cbb         }
433         it = se.insert(p).first;
69e         if (it != se.begin()) mid(*prev(it), *it, 0);
96d         if (next(it) != se.end()) mid(*it, *next(it), 0);
6a5         return 1;

```

```

cbb     }
5eb     int query() {
956         if (!quina.size()) return INF;
add         return *quina.begin();
cbb     }
214 };

```

### 3.16 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

c53 void lis2d(vector<vector<tuple<int, int, int>>>& v,
vector<int>& dp, int l, int r) {
893     if (l == r) {
56f         for (int i = 0; i < v[l].size(); i++) {
8b5             int ii = get<2>(v[l][i]);
1ce             dp[ii] = max(dp[ii], 1);
cbb         }
505         return;
cbb     }
ee4     int m = (l+r)/2;
62b     lis2d(v, dp, l, m);

325     vector<tuple<int, int, int>> vv[2];
d44     vector<int> Z;
871     for (int i = l; i <= r; i++) for (auto it : v[i]) {
2ef         vv[i > m].push_back(it);
042         Z.push_back(get<1>(it));
cbb     }
e9f     sort(vv[0].begin(), vv[0].end());
9b5     sort(vv[1].begin(), vv[1].end());
0d1     sort(Z.begin(), Z.end());
573     auto get_z = [&](int z) { return
lower_bound(Z.begin(), Z.end(), z) - Z.begin(); };
c51     vector<int> bit(Z.size());

```

```

181     int i = 0;
e9a     for (auto [y, z, id] : vv[1]) {
6bd         while (i < vv[0].size() and get<0>(vv[0][i]) <
y) {
397             auto [y2, z2, id2] = vv[0][i++];
ea0             for (int p = get_z(z2)+1; p <= Z.size(); p
+= p&-p)
300                 bit[p-1] = max(bit[p-1], dp[id2]);
cbb         }
d3b         int q = 0;
fd9         for (int p = get_z(z); p; p -= p&-p) q = max(q,
bit[p-1]);
614         dp[id] = max(dp[id], q + 1);
cbb     }
c25     lis2d(v, dp, m+1, r);
cbb }

4de vector<int> solve(vector<tuple<int, int, int>> v) {
3d2     int n = v.size();
cd4     vector<tuple<int, int, int, int>> vv;
603     for (int i = 0; i < n; i++) {
9be         auto [x, y, z] = v[i];
5bb         vv.emplace_back(x, y, z, i);
cbb     }
bd3     sort(vv.begin(), vv.end());

e11     vector<vector<tuple<int, int, int>>> V;
603     for (int i = 0; i < n; i++) {
a5b         int j = i;
808         V.emplace_back();
c01         while (j < n and get<0>(vv[j]) == get<0>(vv[i]))
{
ba6             auto [x, y, z, id] = vv[j++];
cbb             V.back().emplace_back(y, z, id);
cbb         }
452         i = j-1;
cbb     }
388     vector<int> dp(n);
839     lis2d(V, dp, 0, V.size()-1);
898     return dp;
cbb }

```

### 3.17 Gray Code

```

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

df6 vector<int> gray_code(int n) {
73f     vector<int> ret(1<<n);
f29     for (int i = 0; i < (1<<n); i++) ret[i] = i^(i>>1);
edf     return ret;
cbb }

```

### 3.18 Half-plane intersection

```

// Cada half-plane eh identificado por uma reta e a regioao
// ccw a ela
//
// 0(n log n)
// f56e1c

f4f vector<pt> hp_intersection(vector<line> &v) {
9bc     deque<pt> dq = {{INF, INF}, {-INF, INF}, {-INF,
-INF}, {INF, -INF}};

d41 #warning considerar trocar por compare_angle
de3     sort(v.begin(), v.end(), [&](line r, line s) {
return angle(r.q-r.p) < angle(s.q-s.p); });

5e9     for(int i = 0; i < v.size() and dq.size() > 1; i++) {
c69         pt p1 = dq.front(), p2 = dq.back();
6c6         while (dq.size() and !ccw(v[i].p, v[i].q,
dq.back()))
47b             p1 = dq.back(), dq.pop_back();
0a2         while (dq.size() and !ccw(v[i].p, v[i].q,
dq.front()))
7cf             p2 = dq.front(), dq.pop_front();

4d9         if (!dq.size()) break;
606         if (p1 == dq.front() and p2 == dq.back())
continue;

```

```

c9b         dq.push_back(inter(v[i], line(dq.back(), p1)));
65c         dq.push_front(inter(v[i], line(dq.front(), p2)));

fdd         if (dq.size() > 1 and dq.back() == dq.front())
            dq.pop_back();
cbb     }
b2b     return vector<pt>(dq.begin(), dq.end());
cbb }

```

### 3.19 Heap Sort

```

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

f18 void down(vector<int>& v, int n, int i) {
e1f     while ((i = 2*i+1) < n) {
583         if (i+1 < n and v[i] < v[i+1]) i++;
b27         if (v[i] < v[(i-1)/2]) break;
322         swap(v[i], v[(i-1)/2]);
cbb     }
cbb }

eb6 void heap_sort(vector<int>& v) {
3d2     int n = v.size();
61d     for (int i = n/2-1; i >= 0; i--) down(v, n, i);
917     for (int i = n-1; i > 0; i--)
37f         swap(v[0], v[i]), down(v, i, 0);
cbb }

```

### 3.20 Inversion Count

```

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

37b template<typename T> ll inv_count(vector<T> l, vector<T>
    r = {}) {
bb6     if (!r.size()) {
796         r = l;
1bc         sort(r.begin(), r.end());

```

```

cbb     }
874     int n = l.size();
8c0     vector<int> v(n), bit(n);
4e9     vector<pair<T, int>> w;
61c     for (int i = 0; i < n; i++) w.push_back({r[i], i+1});
d1d     sort(w.begin(), w.end());
603     for (int i = 0; i < n; i++) {
bf3         auto it = lower_bound(w.begin(), w.end(),
            make_pair(l[i], 0));
1bf         if (it == w.end() or it->first != l[i]) return
            -1; // nao da
962         v[i] = it->second;
6c0         it->second = -1;
cbb     }

04b     ll ans = 0;
45b     for (int i = n-1; i >= 0; i--) {
2d9         for (int j = v[i]-1; j; j -= j&-j) ans += bit[j];
3a1         for (int j = v[i]; j < n; j += j&-j) bit[j]++;
cbb     }
ba7     return ans;
cbb }

```

### 3.21 LIS - Longest Increasing Subsequence

```

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

121 template<typename T> vector<T> lis(vector<T>& v) {
1fa     int n = v.size(), m = -1;
f0c     vector<T> d(n+1, INF);
aec     vector<int> l(n);
007     d[0] = -INF;

603     for (int i = 0; i < n; i++) {
            // Para non-decreasing use upper_bound()
4fd         int t = lower_bound(d.begin(), d.end(), v[i]) -
            d.begin();
3ad         d[t] = v[i], l[i] = t, m = max(m, t);
cbb     }

```

```

4ff      int p = n;
5a9      vector<T> ret;
cdf      while (p--> if (l[p] == m) {
883          ret.push_back(v[p]);
76b          m--;
cbb      }
969      reverse(ret.begin(), ret.end());

edf      return ret;
cbb }

```

## 3.22 LIS2 - Longest Increasing Subsequence

```

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

84b template<typename T> int lis(vector<T> &v){
2da     vector<T> ans;
5e0     for (T t : v){
        // Para non-decreasing use upper_bound()
fe6         auto it = lower_bound(ans.begin(), ans.end(), t);
d7f         if (it == ans.end()) ans.push_back(t);
b94         else *it = t;
cbb     }
1eb     return ans.size();
cbb }

```

## 3.23 Minimum Enclosing Circle

```

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

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

b2a struct pt {
662     double x, y;

```

```

be7     pt(double x_ = 0, double y_ = 0) : x(x_), y(y_) {}
7af     pt operator + (const pt& p) const { return pt(x+p.x,
y+p.y); }
b23     pt operator - (const pt& p) const { return pt(x-p.x,
y-p.y); }
254     pt operator * (double c) const { return pt(x*c,
y*c); }
701     pt operator / (double c) const { return pt(x/c,
y/c); }
214 };

```

```

2f9 double dot(pt p, pt q) { return p.x*q.x+p.y*q.y; }
dd5 double cross(pt p, pt q) { return p.x*q.y-p.y*q.x; }
e7c double dist(pt p, pt q) { return sqrt(dot(p-q, p-q)); }

```

```

3f4 pt center(pt p, pt q, pt r) {
5d9     pt a = p-r, b = q-r;
e84     pt c = pt(dot(a, p+r)/2, dot(b, q+r)/2);
e01     return pt(cross(c, pt(a.y, b.y)), cross(pt(a.x,
b.x), c)) / cross(a, b);
cbb }

```

```

aa8 struct circle {
f41     pt cen;
c12     double r;
898     circle(pt cen_, double r_) : cen(cen_), r(r_) {}
83c     circle(pt a, pt b, pt c) {
13d         cen = center(a, b, c);
1f1         r = dist(cen, a);
cbb     }
cd5     bool inside(pt p) { return dist(p, cen) < r+EPS; }
214 };

```

```

806 circle minCirc(vector<pt> v) {
f21     shuffle(v.begin(), v.end(), rng);
ae0     circle ret = circle(pt(0, 0), 0);
618     for (int i = 0; i < v.size(); i++) if
        (!ret.inside(v[i])) {
16a         ret = circle(v[i], 0);
f11         for (int j = 0; j < i; j++) if
            (!ret.inside(v[j])) {
881             ret = circle((v[i]+v[j])/2, dist(v[i],

```

```

        v[j])/2);
b8c         for (int k = 0; k < j; k++) if
            (!ret.inside(v[k]))
43f             ret = circle(v[i], v[j], v[k]);
cbb         }
cbb     }
edf     return ret;
cbb }

```

### 3.24 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
539 vector<pt> minkowski(vector<pt> p, vector<pt> q) {
051     auto fix = [](vector<pt>& P) {
515         rotate(P.begin(), min_element(P.begin(),
P.end()), P.end());
018         P.push_back(P[0]), P.push_back(P[1]);
214     };
889     fix(p), fix(q);
8af     vector<pt> ret;
692     int i = 0, j = 0;
2ee     while (i < p.size()-2 or j < q.size()-2) {
898         ret.push_back(p[i] + q[j]);
732         auto c = ((p[i+1] - p[i]) ^ (q[j+1] - q[j]));
ebc         if (c >= 0) i = min<int>(i+1, p.size()-2);
81e         if (c <= 0) j = min<int>(j+1, q.size()-2);
cbb     }
edf     return ret;
cbb }

// 2f5dd2
c3e ld dist_convex(vector<pt> p, vector<pt> q) {
dc2     for (pt& i : p) i = i * -1;
44c     auto s = minkowski(p, q);
95d     if (inpol(s, pt(0, 0))) return 0;
6a5     return 1;

```

```

921     ld ans = DINF;
073     for (int i = 0; i < s.size(); i++) ans = min(ans,
f04         disttoseg(pt(0, 0), line(s[(i+1)%s.size()],
s[i])));
ba7     return ans;
cbb }

```

### 3.25 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))
// f98540

8d3 struct dsu {
553     int n, ans;
2e3     vector<int> p, sz;
ee6     stack<int> S;

4b8     dsu(int n_) : n(n_), ans(n), p(n), sz(n) {
8a6         for (int i = 0; i < n; i++) p[i] = i, sz[i] = 1;
cbb     }
1b1     int find(int k) {
006         while (p[k] != k) k = p[k];
839         return k;
cbb     }
553     void add(pair<int, int> x) {
700         int a = x.first, b = x.second;
605         a = find(a), b = find(b);
843         if (a == b) return S.push(-1);
e7d         ans--;
3c6         if (sz[a] > sz[b]) swap(a, b);
4c2         S.push(a);
582         sz[b] += sz[a];
84b         p[a] = b;
cbb     }
35c     int query() { return ans; }
5cf     void rollback() {

```

```

465         int u = S.top(); S.pop();
61c         if (u == -1) return;
270         sz[p[u]] -= sz[u];
546         p[u] = u;
0df         ans++;
cbb     }
214 };

1a8 int n;
e93 vector<pair<int, int>> ar; // vetor com as arestas

617 vector<int> M0(vector<pair<int, int>> &q) {
d4d     int SQ = ar.size() / sqrt(q.size()) + 1;
c23     int m = q.size();
3f8     vector<int> ord(m);
be8     iota(ord.begin(), ord.end(), 0);
d01     sort(ord.begin(), ord.end(), [&](int l, int r) {
9c9         if (q[l].first / SQ != q[r].first / SQ) return
q[l].first < q[r].first;
a66         return q[l].second < q[r].second;
c0c     });
435     vector<int> ret(m);

dd5     for (int i = 0; i < m; i++) {
176         dsu D(n);
ae9         int fim = q[ord[i]].first/SQ*SQ + SQ - 1;
e25         int last_r = fim;
ebc         int j = i-1;
00c         while (j+1 < m and q[ord[j+1]].first / SQ ==
q[ord[i]].first / SQ) {
a0e             auto [l, r] = q[ord[++j]];

acc             if (l / SQ == r / SQ) {
ce9                 dsu D2(n);
495                 for (int k = 1; k <= r; k++)
D2.add(ar[k]);
fdf                 ret[ord[j]] = D2.query();
5e2                 continue;
cbb             }

59b             while (last_r < r) D.add(ar[++last_r]);
2cf             for (int k = 1; k <= fim; k++) D.add(ar[k]);

```

```

9b2             ret[ord[j]] = D.query();

572             for (int k = 1; k <= fim; k++) D.rollback();
cbb         }
bdf         i = j;
cbb     }
edf     return ret;
cbb }

```

### 3.26 Mo - numero de distintos em range

```

// Para ter o bound abaixo, escolher
// SQ = n / sqrt(q)
//
// O(n * sqrt(q))
// e94f60

```

```

0d2 const int MAX = 1e5+10;
6ff const int SQ = sqrt(MAX);
b69 int v[MAX];

```

```

b65 int ans, freq[MAX];

```

```

9da inline void insert(int p) {
ae0     int o = v[p];
591     freq[o]++;
992     ans += (freq[o] == 1);
cbb }

```

```

a25 inline void erase(int p) {
ae0     int o = v[p];
7ee     ans -= (freq[o] == 1);
ba2     freq[o]--;
cbb }

```

```

e51 inline ll hilbert(int x, int y) {
71e     static int N = 1 << (__builtin_clz(0) -
__builtin_clz(MAX));
100     int rx, ry, s;
b72     ll d = 0;
43b     for (s = N/2; s > 0; s /= 2) {

```

```

c95         rx = (x & s) > 0, ry = (y & s) > 0;
e3e         d += s * ll(s) * ((3 * rx) ^ ry);
d2e         if (ry == 0) {
5aa             if (rx == 1) x = N-1 - x, y = N-1 - y;
9dd             swap(x, y);
cbb         }
cbb     }
be2     return d;
cbb }

bac #define HILBERT true
617 vector<int> M0(vector<pair<int, int>> &q) {
c3b     ans = 0;
c23     int m = q.size();
3f8     vector<int> ord(m);
be8     iota(ord.begin(), ord.end(), 0);
6a6 #if HILBERT
8c4     vector<ll> h(m);
74c     for (int i = 0; i < m; i++) h[i] =
        hilbert(q[i].first, q[i].second);
075     sort(ord.begin(), ord.end(), [&](int l, int r) {
        return h[l] < h[r]; });
8c1 #else
d01     sort(ord.begin(), ord.end(), [&](int l, int r) {
9c9         if (q[l].first / SQ != q[r].first / SQ) return
            q[l].first < q[r].first;
0db         if ((q[l].first / SQ) % 2) return q[l].second >
            q[r].second;
a66         return q[l].second < q[r].second;
c0c     });
f2e #endif
435     vector<int> ret(m);
3d9     int l = 0, r = -1;

8b0     for (int i : ord) {
6c6         int ql, qr;
4f5         tie(ql, qr) = q[i];
026         while (r < qr) insert(++r);
232         while (l > ql) insert(--l);
75e         while (l < ql) erase(l++);
fe8         while (r > qr) erase(r--);
381         ret[i] = ans;

```

```

cbb     }
edf     return ret;
cbb }

```

## 3.27 Palindromic Factorization

```

// Precisa da eertree
// Computa o numero de formas de particionar cada
// prefixo da string em strings palindromicas
//
// 0(n log n), considerando alfabeto 0(1)
// 9e6e22

070 struct eertree { ... };

0e7 ll factorization(string s) {
b19     int n = s.size(), sz = 2;
580     eertree PT(n);
147     vector<int> diff(n+2), slink(n+2), sans(n+2),
        dp(n+1);
0ec     dp[0] = 1;
78a     for (int i = 1; i <= n; i++) {
c58         PT.add(s[i-1]);
a7c         if (PT.size()+2 > sz) {
6c4             diff[sz] = PT.len[sz] - PT.len[PT.link[sz]];
241             if (diff[sz] == diff[PT.link[sz]])
d6f                 slink[sz] = slink[PT.link[sz]];
f53             else slink[sz] = PT.link[sz];
eb9             sz++;
cbb         }
911         for (int v = PT.last; PT.len[v] > 0; v =
            slink[v]) {
297             sans[v] = dp[i - (PT.len[slink[v]] +
                diff[v])];
85d             if (diff[v] == diff[PT.link[v]])
f20                 sans[v] = (sans[v] + sans[PT.link[v]]) %
                    MOD;
071             dp[i] = (dp[i] + sans[v]) % MOD;
cbb         }
cbb     }
5f0     return dp[n];
cbb }

```

## 3.28 Parsing de Expressao

```
// Operacoes associativas a esquerda por default
// Para mudar isso, colocar em r_assoc
// Operacoes com maior prioridade sao feitas primeiro
//
// 68921b

cc1 bool blank(char c) {
f34     return c == ' ';
cbb }

8e4 bool is_unary(char c) {
f9c     return c == '+' or c == '-';
cbb }

76d bool is_op(char c) {
010     if (is_unary(c)) return true;
31c     return c == '*' or c == '/' or c == '+' or c == '-';
cbb }

fa3 bool r_assoc(char op) {
        // operator unario - deve ser assoc. a direita
cf0     return op < 0;
cbb }

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

727     if (op == '*' or op == '/') return 2;
439     if (op == '+' or op == '-') return 1;
daa     return -1;
cbb }

c15 void process_op(stack<int>& st, stack<int>& op) {
88c     char o = op.top(); op.pop();
91c     if (o < 0) {
4e6         o *= -1;
1e2         int l = st.top(); st.pop();
0ff         if (o == '+') st.push(l);
7e9         if (o == '-') st.push(-l);
```

```
9d9     } else {
14c         int r = st.top(); st.pop();
1e2         int l = st.top(); st.pop();
1e4         if (o == '*') st.push(l * r);
f55         if (o == '/') st.push(l / r);
605         if (o == '+') st.push(l + r);
c40         if (o == '-') st.push(l - r);
cbb     }
cbb }

439 int eval(string& s) {
212     stack<int> st, op;
d0c     bool un = true;
1cf     for (int i = 0; i < s.size(); i++) {
68d         if (blank(s[i])) continue;

139         if (s[i] == '(') {
367             op.push('(');
99d             un = true;
130         } else if (s[i] == ')') {
709             while (op.top() != '(') process_op(st, op);
75e             op.pop();
ce2             un = false;
146         } else if (is_op(s[i])) {
4d0             char o = s[i];
37c             if (un and is_unary(o)) o *= -1;
ae3             while (op.size() and (
cd6                 (!r_assoc(o) and
                    priority(op.top()) >= priority(o)) or
c41                 (r_assoc(o) and
                    priority(op.top()) > priority(o))))
c47                 process_op(st, op);
c00             op.push(o);
99d             un = true;
9d9         } else {
da8             int val = 0;
c2b             while (i < s.size() and isalnum(s[i]))
8a3                 val = val * 10 + s[i++] - '0';
169             i--;
25d             st.push(val);
ce2             un = false;
cbb         }
```



```

cbb      }

7f6      while (op.size()) process_op(st, op);
123      return st.top();
cbb }

```

### 3.29 RMQ com Divide and Conquer

```

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

f74 typedef pair<pair<int, int>, int> iii;
7c6 #define f first
0ab #define s second

87d int n, q, v[MAX];
e3f iii qu[MAX];
aeb int ans[MAX], pref[MAX], sulf[MAX];

0e3 void solve(int l=0, int r=n-1, int ql=0, int qr=q-1) {
8a3     if (l > r or ql > qr) return;
ee4     int m = (l+r)/2;
1b1     int qL = partition(qu+ql, qu+qr+1, [=](iii x){return
x.f.s < m;}) - qu;
eb0     int qR = partition(qu+qL, qu+qr+1, [=](iii x){return
x.f.f <=m;}) - qu;

3cd     pref[m] = sulf[m] = v[m];
9f9     for (int i = m-1; i >= l; i--) pref[i] = min(v[i],
pref[i+1]);
ea8     for (int i = m+1; i <= r; i++) sulf[i] = min(v[i],
sulf[i-1]);

b2a     for (int i = qL; i < qR; i++)
f3a         ans[qu[i].s] = min(pref[qu[i].f.f],
sulf[qu[i].f.s]);

364     solve(l, m-1, ql, qL-1), solve(m+1, r, qR, qr);
cbb }

```

### 3.30 Segment Intersection

```

// Verifica, dado n segmentos, se existe algum par de
// segmentos
// que se intersecta
//
// O(n log n)
// 3957d8

6e0 bool operator < (const line& a, const line& b) { //
comparador pro sweepline
191     if (a.p == b.p) return ccw(a.p, a.q, b.q);
231     if (!eq(a.p.x, a.q.x) and (eq(b.p.x, b.q.x) or
a.p.x+eps < b.p.x))
780         return ccw(a.p, a.q, b.p);
dc0     return ccw(a.p, b.q, b.p);
cbb }

8e2 bool has_intersection(vector<line> v) {
576     auto intersects = [&](pair<line, int> a, pair<line,
int> b) {
a08         return interseg(a.first, b.first);
214     };
e1b     vector<pair<pt, pair<int, int>>> w;
f14     for (int i = 0; i < v.size(); i++) {
876         if (v[i].q < v[i].p) swap(v[i].p, v[i].q);
e1d         w.push_back({v[i].p, {0, i}});
034         w.push_back({v[i].q, {1, i}});
cbb     }
d1d     sort(w.begin(), w.end());
7f2     set<pair<line, int>> se;
e58     for (auto i : w) {
bfd         line at = v[i].second.second;
292         if (i.second.first == 0) {
145             auto nxt = se.lower_bound({at,
i.second.second});
d1e             if (nxt != se.end() and intersects(*nxt,
{at, i.second.second})) return 1;
257             if (nxt != se.begin() and
intersects(*(--nxt), {at, i.second.second})) return 1;
78f             se.insert({at, i.second.second});
9d9         } else {

```

```

884         auto nxt = se.upper_bound({at,
i.second.second}), cur = nxt, prev = --cur;
b64         if (nxt != se.end() and prev != se.begin()
4fb             and intersects(*nxt, *(--prev))) return
1;
cca         se.erase(cur);
cbb     }
cbb     }
bb3     return 0;
cbb }

```

### 3.31 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

```

```

860 vector<int> de_bruijn(int n, int k, int lim = INF) {
b55     if (k == 1) return vector<int>(lim == INF ? 1 : n,
0);
5f6     vector<int> l = {0}, ret; // l eh lyndon word
667     while (true) {
c86         if (l.size() == 0) {
1b9             if (lim == INF) break;
daf             l.push_back(0);
cbb         }
686         if (n % l.size() == 0) for (int i : l) {
728             ret.push_back(i);
c99             if (ret.size() == n+lim-1) return ret;
cbb         }
630         int p = l.size();
905         while (l.size() < n) l.push_back(l[l.size()%p]);
e7f         while (l.size() and l.back() == k-1)

```

```

l.pop_back();
88a         if (l.size()) l.back()++;
cbb     }
edf     return ret;
cbb }

```

### 3.32 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)
3de pair<vector<int>, vector<int>> addition_chain() {
16f     int MAX = 301;
875     vector<int> dp(MAX), p(MAX);
1ab     for (int n = 2; n < MAX; n++) {
7c0         pair<int, int> val = {INF, -1};
212         for (int i = 1; i < n; i++) for (int j = i; j; j
= p[j])
94a             if (j == n-i) val = min(val, pair(dp[i]+1,
i));
eb3         tie(dp[n], p[n]) = val;
efe         if (n == 9) p[n] = 8;
ba1         if (n == 149 or n == 233) dp[n]--;
cbb     }
717     return {dp, p};
cbb }

```

### 3.33 Simple Polygon

```

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

```

```

// c724a4

6e0 bool operator < (const line& a, const line& b) { //
    comparador pro sweepline
191     if (a.p == b.p) return ccw(a.p, a.q, b.q);
231     if (!eq(a.p.x, a.q.x) and (eq(b.p.x, b.q.x) or
        a.p.x+eps < b.p.x))
780         return ccw(a.p, a.q, b.p);
dc0     return ccw(a.p, b.q, b.p);
cbb }

6f3 bool simple(vector<pt> v) {
576     auto intersects = [&](pair<line, int> a, pair<line,
    int> b) {
e72         if ((a.second+1)%v.size() == b.second or
80e         (b.second+1)%v.size() == a.second) return
        false;
a08         return interseg(a.first, b.first);
214     };
41a     vector<line> seg;
e1b     vector<pair<pt, pair<int, int>>> w;
f14     for (int i = 0; i < v.size(); i++) {
0a8         pt at = v[i], nxt = v[(i+1)%v.size()];
828         if (nxt < at) swap(at, nxt);
937         seg.push_back(line(at, nxt));
f7e         w.push_back({at, {0, i}});
69c         w.push_back({nxt, {1, i}});
        // casos degenerados estranhos
ae8         if (isinseg(v[(i+2)%v.size()], line(at, nxt)))
            return 0;
88d         if (isinseg(v[(i+v.size()-1)%v.size()], line(at,
            nxt))) return 0;
cbb     }
d1d     sort(w.begin(), w.end());
7f2     set<pair<line, int>> se;
e58     for (auto i : w) {
ff8         line at = seg[i.second.second];
292         if (i.second.first == 0) {
145             auto nxt = se.lower_bound({at,
                i.second.second});
7c4             if (nxt != se.end() and intersects(*nxt,
                {at, i.second.second})) return 0;

```

```

b34         if (nxt != se.begin() and
            intersects(*(--nxt), {at, i.second.second})) return 0;
78f         se.insert({at, i.second.second});
9d9     } else {
884         auto nxt = se.upper_bound({at,
            i.second.second}), cur = nxt, prev = --cur;
b64         if (nxt != se.end() and prev != se.begin()
403         and intersects(*nxt, *(--prev))) return
            0;
cca         se.erase(cur);
cbb     }
cbb }
6a5     return 1;
cbb }

```

### 3.34 Sweep Direction

```

// Passa por todas as ordenacoes dos pontos definidas por
    "direcoes"
// Assume que nao existem pontos coincidentes
//
// 0(n^2 log n)
// 6bb68d

4b8 void sweep_direction(vector<pt> v) {
3d2     int n = v.size();
163     sort(v.begin(), v.end(), [](pt a, pt b) {
3a5         if (a.x != b.x) return a.x < b.x;
572         return a.y > b.y;
c0c     });
b89     vector<int> at(n);
516     iota(at.begin(), at.end(), 0);
b79     vector<pair<int, int>> swapp;
25e     for (int i = 0; i < n; i++) for (int j = i+1; j < n;
j++)
95f         swapp.push_back({i, j}), swapp.push_back({j, i});

269     sort(swapp.begin(), swapp.end(), [&](auto a, auto b)
{
134         pt A = rotate90(v[a.first] - v[a.second]);
247         pt B = rotate90(v[b.first] - v[b.second]);
615         if (quad(A) == quad(B) and !sarea2(pt(0, 0), A,

```

```

B)) return a < b;
224     return compare_angle(A, B);
c0c     });
4e6     for (auto par : swapp) {
e24         assert(abs(at[par.first] - at[par.second]) == 1);
a96         int l = min(at[par.first], at[par.second]),
0d3             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)
9cf         swap(v[at[par.first]], v[at[par.second]]);
1c0         swap(at[par.first], at[par.second]);
cbb     }
cbb }

```

### 3.35 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)
// 83ebab

```

```

2ad typedef struct QuadEdge* Q;
ba5 struct QuadEdge {
53e     int id;
114     pt o;
41e     Q rot, nxt;
3e5     bool used;

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

00f     Q rev() const { return rot->rot; }
c3c     Q next() const { return nxt; }
188     Q prev() const { return rot->next()->rot; }
0d4     pt dest() const { return rev()->o; }
214 };

91b Q edge(pt from, pt to, int id_from, int id_to) {
c6e     Q e1 = new QuadEdge(id_from, from);
61b     Q e2 = new QuadEdge(id_to, to);
8f6     Q e3 = new QuadEdge;
5ca     Q e4 = new QuadEdge;
e69     tie(e1->rot, e2->rot, e3->rot, e4->rot) = {e3, e4,
        e2, e1};
f22     tie(e1->nxt, e2->nxt, e3->nxt, e4->nxt) = {e1, e2,
        e4, e3};
1ad     return e1;
cbb }

d8d void splice(Q a, Q b) {
a6f     swap(a->nxt->rot->nxt, b->nxt->rot->nxt);
da4     swap(a->nxt, b->nxt);
cbb }

167 void del_edge(Q& e, Q ne) { // delete e and assign e <-
    ne
cc0     splice(e, e->prev());
eec     splice(e->rev(), e->rev()->prev());
7ea     delete e->rev()->rot, delete e->rev();
524     delete e->rot; delete e;
6b2     e = ne;

```

```

cbb }

d08 Q conn(Q a, Q b) {
cc5     Q e = edge(a->dest(), b->o, a->rev()->id, b->id);
f2b     splice(e, a->rev()->prev());
d37     splice(e->rev(), b);
6bf     return e;
cbb }

d64 bool in_c(pt a, pt b, pt c, pt p) { // p ta na circumf.
(a, b, c) ?
268     __int128 p2 = p*p, A = a*a - p2, B = b*b - p2, C =
c*c - p2;
cbe     return sarea2(p, a, b) * C + sarea2(p, b, c) * A +
sarea2(p, c, a) * B > 0;
cbb }

540 pair<Q, Q> build_tr(vector<pt>& p, int l, int r) {
09d     if (r-l+1 <= 3) {
2eb         Q a = edge(p[l], p[l+1], l, l+1), b =
edge(p[l+1], p[r], l+1, r);
912         if (r-l+1 == 2) return {a, a->rev()};
0ec         splice(a->rev(), b);
c3c         ll ar = sarea2(p[l], p[l+1], p[r]);
1af         Q c = ar ? conn(b, a) : 0;
021         if (ar >= 0) return {a, b->rev()};
9db         return {c->rev(), c};
cbb     }
ee4     int m = (l+r)/2;
328     auto [la, ra] = build_tr(p, l, m);
b93     auto [lb, rb] = build_tr(p, m+1, r);
667     while (true) {
b99         if (ccw(lb->o, ra->o, ra->dest())) ra =
ra->rev()->prev();
458         else if (ccw(lb->o, ra->o, lb->dest())) lb =
lb->rev()->next();
f97         else break;
cbb     }
ca5     Q b = conn(lb->rev(), ra);
713     auto valid = [&](Q e) { return ccw(e->dest(),
b->dest(), b->o); };
ee1     if (ra->o == la->o) la = b->rev();

```

```

63f         if (lb->o == rb->o) rb = b;
667         while (true) {
71e             Q L = b->rev()->next();
d11             if (valid(L)) while (in_c(b->dest(), b->o,
L->dest(), L->next()->dest()))
1c0                 del_edge(L, L->next());
c76             Q R = b->prev();
2b0             if (valid(R)) while (in_c(b->dest(), b->o,
R->dest(), R->prev()->dest()))
541                 del_edge(R, R->prev());
a3a             if (!valid(L) and !valid(R)) break;
ccd             if (!valid(L) or (valid(R) and in_c(L->dest(),
L->o, R->o, R->dest()))))
36c                 b = conn(R, b->rev());
666             else b = conn(b->rev(), L->rev());
cbb         }
a2b         return {la, rb};
cbb     }

b58 vector<vector<int>> delaunay(vector<pt> v) {
3d2     int n = v.size();
397     auto tmp = v;
135     vector<int> idx(n);
295     iota(idx.begin(), idx.end(), 0);
fe9     sort(idx.begin(), idx.end(), [&](int l, int r) {
return v[l] < v[r]; });
5d8     for (int i = 0; i < n; i++) v[i] = tmp[idx[i]];
780     assert(unique(v.begin(), v.end()) == v.end());
4aa     vector<vector<int>> g(n);
4ec     bool col = true;
a96     for (int i = 2; i < n; i++) if (sarea2(v[i], v[i-1],
v[i-2])) col = false;
bf5     if (col) {
aa4         for (int i = 1; i < n; i++)
839             g[idx[i-1]].push_back(idx[i]),
g[idx[i]].push_back(idx[i-1]);
96b         return g;
cbb     }
d36     Q e = build_tr(v, 0, n-1).first;
113     vector<Q> edg = {e};
5d1     for (int i = 0; i < edg.size(); e = edg[i++]) {
3ed         for (Q at = e; !at->used; at = at->next()) {

```

```

60d         at->used = true;
cf8         g[idx[at->id]].push_back(idx[at->rev()->id]);
15d         edg.push_back(at->rev());
cbb     }
cbb     }
96b     return g;
cbb }

```

### 3.36 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

```

```

c0d vector<pair<int, int>> g[MAX]; // {para, peso}

d41 #warning o 'g' deve estar ordenado
9a5 vector<pair<int, pair<int, int>>> get_triangles(int i) {
771     vector<pair<int, pair<int, int>>> tri;
b23     for (pair<int, int> j : g[i]) {
2b3         int a = i, b = j.first;
6dd         if (g[a].size() > g[b].size()) swap(a, b);
eb0         for (pair<int, int> c : g[a]) if (c.first != b
and c.first > j.first) {
525             auto it = lower_bound(g[b].begin(),
g[b].end(), make_pair(c.first, -INF));
f55             if (it == g[b].end() or it->first !=
c.first) continue;
0aa             tri.push_back({j.second+c.second+it->second,
{a == i ? b : a, c.first}});
cbb         }
cbb     }
f5e     return tri;
cbb }

```

## 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
//
//  $O(|V|+|E|) = O(\text{\#variaveis} + \text{\#restricoes})$ 
// ef6b3b

```

```

138 struct sat {
e6c     int n, tot;
789     vector<vector<int>> g;
0ca     vector<int> vis, comp, id, ans;
4ce     stack<int> s;

141     sat() {}
172     sat(int n_) : n(n_), tot(n), g(2*n) {}

f32     int dfs(int i, int& t) {
cf0         int lo = id[i] = t++;
efc         s.push(i), vis[i] = 2;
48e         for (int j : g[i]) {
740             if (!vis[j]) lo = min(lo, dfs(j, t));
994             else if (vis[j] == 2) lo = min(lo, id[j]);
cbb         }
3de         if (lo == id[i]) while (1) {
3c3             int u = s.top(); s.pop();
9c5             vis[u] = 1, comp[u] = i;
91d             if ((u>>1) < n and ans[u>>1] == -1)
ans[u>>1] = ~u&1;
2ef                 if (u == i) break;
cbb             }
253             return lo;
cbb         }

74a     void add_impl(int x, int y) { // x -> y = !x ou y
26a         x = x >= 0 ? 2*x : -2*x-1;
2b8         y = y >= 0 ? 2*y : -2*y-1;
a1e         g[x].push_back(y);
1e2         g[y^1].push_back(x^1);

```

```

cbb    }
e85    void add_cl(int x, int y) { // x ou y
0b5        add_impl(~x, y);
cbb    }
487    void add_xor(int x, int y) { // x xor y
0b7        add_cl(x, y), add_cl(~x, ~y);
cbb    }
978    void add_eq(int x, int y) { // x = y
c86        add_xor(~x, y);
cbb    }
b10    void add_true(int x) { // x = T
18b        add_impl(~x, x);
cbb    }
d14    void at_most_one(vector<int> v) { // no max um
verdadeiro
54d        g.resize(2*(tot+v.size()));
f14        for (int i = 0; i < v.size(); i++) {
8c9            add_impl(tot+i, ~v[i]);
a8f            if (i) {
b6a                add_impl(tot+i, tot+i-1);
3d3                add_impl(v[i], tot+i-1);
cbb            }
cbb        }
258        tot += v.size();
cbb    }

a8e    pair<bool, vector<int>> solve() {
27b        ans = vector<int>(n, -1);
6bb        int t = 0;
0de        vis = comp = id = vector<int>(2*tot, 0);
53c        for (int i = 0; i < 2*tot; i++) if (!vis[i])
dfs(i, t);
f88        for (int i = 0; i < tot; i++)
4c9            if (comp[2*i] == comp[2*i+1]) return {false,
{} };
997        return {true, ans};
cbb    }
214 };

```

## 4.2 Algoritmo de Euclides estendido

// Acha x e y tal que  $ax + by = \text{mdc}(a, b)$  (nao eh unico)

```

// Assume a, b >= 0
//
// 0(log(min(a, b)))
// 35411d

```

```

2be tuple<ll, ll, ll> ext_gcd(ll a, ll b) {
3bd     if (!a) return {b, 0, 1};
550     auto [g, x, y] = ext_gcd(b%a, a);
c59     return {g, y - b/a*x, x};
cbb }

```

## 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

```

```

ee8 mint evaluate_interpolation(int x, vector<mint> y) {
80e     int n = y.size();

184     vector<mint> sulf(n+1, 1), fat(n, 1), ifat(n);
6fa     for (int i = n-1; i >= 0; i--) sulf[i] = sulf[i+1] *
(x - i);
29b     for (int i = 1; i < n; i++) fat[i] = fat[i-1] * i;
0da     ifat[n-1] = 1/fat[n-1];
3db     for (int i = n-2; i >= 0; i--) ifat[i] = ifat[i+1] *
(i + 1);

ca1     mint pref = 1, ans = 0;
5ea     for (int i = 0; i < n; pref *= (x - i++)) {
42f         mint num = pref * sulf[i+1];

b4e         mint den = ifat[i] * ifat[n-1 - i];
0bd         if ((n-1 - i)%2) den *= -1;

03f         ans += y[i] * num * den;
cbb     }
ba7     return ans;

```

```
cbb }
```

## 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
//
// 0(n^2 log k), em que n = |s|
// 8644e3

b7c template<typename T> T evaluate(vector<T> c, vector<T>
    s, ll k) {
ff2     int n = c.size();
9ee     assert(c.size() <= s.size());

d09     auto mul = [&](const vector<T> &a, const vector<T>
    &b) {
564         vector<T> ret(a.size() + b.size() - 1);
d75         for (int i = 0; i < a.size(); i++) for (int j =
    0; j < b.size(); j++)
cff             ret[i+j] += a[i] * b[j];
83d         for (int i = ret.size()-1; i >= n; i--) for (int
    j = n-1; j >= 0; j--)
112             ret[i-j-1] += ret[i] * c[j];
16d         ret.resize(min<int>(ret.size(), n));
edf         return ret;
214     };

1a6     vector<T> a = n == 1 ? vector<T>({c[0]}) :
vector<T>({0, 1}), x = {1};
95f     while (k) {
7f1         if (k&1) x = mul(x, a);
b28         a = mul(a, a), k >>= 1;
cbb     }
dd6     x.resize(n);

ce8     T ret = 0;
e72     for (int i = 0; i < n; i++) ret += x[i] * s[i];
edf     return ret;
```

```
cbb }
```

```
192 template<typename T> vector<T>
    berlekamp_massey(vector<T> s) {
ce8     int n = s.size(), l = 0, m = 1;
222     vector<T> b(n), c(n);
46e     T ld = b[0] = c[0] = 1;
620     for (int i = 0; i < n; i++, m++) {
793         T d = s[i];
ab6         for (int j = 1; j <= l; j++) d += c[j] * s[i-j];
5f0         if (d == 0) continue;
8b4         vector<T> temp = c;
369         T coef = d / ld;
ba6         for (int j = m; j < n; j++) c[j] -= coef *
            b[j-m];
88f         if (2 * l <= i) l = i + 1 - l, b = temp, ld = d,
            m = 0;
cbb     }
90c     c.resize(l + 1);
844     c.erase(c.begin());
0dc     for (T& x : c) x = -x;
807     return c;
cbb }

2cf template<typename T> T guess_kth(const vector<T>& s, ll
    k) {
cc3     auto c = berlekamp_massey(s);
96a     return evaluate(c, s, k);
cbb }
```

## 4.5 Binomial Distribution

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

```
361 double logfact[MAX];
```

```
9e4 void calc() {
7a0     logfact[0] = 0;
152     for (int i = 1; i < MAX; i++) logfact[i] =
        logfact[i-1] + log(i);
```



```

cbb }

94c double binom(int n, int k, double p) {
271     return exp(logfact[n] - logfact[k] - logfact[n-k] +
    k * log(p) + (n-k) * log(1 - p));
cbb }

```

## 4.6 Convolucao de GCD / LCM

```

// O(n log(n))

// multiple_transform(a)[i] = \sum_d a[d * i]
// 338be8
bbe template<typename T> void multiple_transform(vector<T>&
    v, bool inv = false) {
64a     vector<int> I(v.size()-1);
847     iota(I.begin(), I.end(), 1);
674     if (inv) reverse(I.begin(), I.end());
dad     for (int i : I) for (int j = 2; i*j < v.size(); j++)
a8a         v[i] += (inv ? -1 : 1) * v[i*j];
cbb }

// gcd_convolution(a, b)[k] = \sum_{gcd(i, j) = k} a_i *
    b_j
// 984f53
fe2 template<typename T> vector<T> gcd_convolution(vector<T>
    a, vector<T> b) {
bdf     multiple_transform(a), multiple_transform(b);
799     for (int i = 0; i < a.size(); i++) a[i] *= b[i];
dea     multiple_transform(a, true);
3f5     return a;
cbb }

// divisor_transform(a)[i] = \sum_{d|i} a[i/d]
// aa74e5
be7 template<typename T> void divisor_transform(vector<T>&
    v, bool inv = false) {
64a     vector<int> I(v.size()-1);
847     iota(I.begin(), I.end(), 1);
5ea     if (!inv) reverse(I.begin(), I.end());
dad     for (int i : I) for (int j = 2; i*j < v.size(); j++)
14f         v[i*j] += (inv ? -1 : 1) * v[i];

```

```

cbb }

// lcm_convolution(a, b)[k] = \sum_{lcm(i, j) = k} a_i *
    b_j
// f5acc1
b1b template<typename T> vector<T> lcm_convolution(vector<T>
    a, vector<T> b) {
3af     divisor_transform(a), divisor_transform(b);
799     for (int i = 0; i < a.size(); i++) a[i] *= b[i];
d8f     divisor_transform(a, true);
3f5     return a;
cbb }

```

## 4.7 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

```

```

58d pair<ll, ll> find_cycle() {
273     ll tort = f(f0);
b2b     ll hare = f(f(f0));
b1b     ll t = 0;
683     while (tort != hare) {
b4d         tort = f(tort);
4b2         hare = f(f(hare));
c82         t++;
cbb     }
0e8     ll st = 0;
909     tort = f0;
683     while (tort != hare) {
b4d         tort = f(tort);
1a2         hare = f(hare);
397         st++;
cbb     }

73d     ll len = 1;
3cd     hare = f(tort);
683     while (tort != hare) {
1a2         hare = f(hare);

```

```

040         len++;
cbb     }
ebd     return {st, len};
cbb }

```

## 4.8 Division Trick

```

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

79c for(int l = 1, r; l <= n; l = r + 1) {
746     r = n / (n / l);
        // n / i has the same value for l <= i <= r
cbb }

```

## 4.9 Eliminacao Gaussiana

```

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

67a template<typename T>
728 pair<int, vector<T>> gauss(vector<vector<T>> a,
    vector<T> b) {
6ca     const double eps = 1e-6;
f92     int n = a.size(), m = a[0].size();
2f0     for (int i = 0; i < n; i++) a[i].push_back(b[i]);

3cb     vector<int> where(m, -1);
237     for (int col = 0, row = 0; col < m and row < n;
        col++) {
f05         int sel = row;
b95         for (int i=row; i<n; ++i)
e55             if (abs(a[i][col]) > abs(a[sel][col])) sel =
i;
2c4         if (abs(a[sel][col]) < eps) continue;
1ae         for (int i = col; i <= m; i++)
dd2             swap(a[sel][i], a[row][i]);

```

```

2c3         where[col] = row;

0c0         for (int i = 0; i < n; i++) if (i != row) {
96c             T c = a[i][col] / a[row][col];
d5c             for (int j = col; j <= m; j++)
c8f                 a[i][j] -= a[row][j] * c;
cbb             }
b70         row++;
cbb     }

```

```

b1d     vector<T> ans(m, 0);
e1a     for (int i = 0; i < m; i++) if (where[i] != -1)
12a         ans[i] = a[where[i]][m] / a[where[i]][i];
603     for (int i = 0; i < n; i++) {
501         T sum = 0;
a75         for (int j = 0; j < m; j++)
5a9             sum += ans[j] * a[i][j];
b1f         if (abs(sum - a[i][m]) > eps)
6cd             return pair(0, vector<T>());
cbb     }

```

```

12e     for (int i = 0; i < m; i++) if (where[i] == -1)
018         return pair(INF, ans);
280     return pair(1, ans);
cbb }

```

## 4.10 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

2a3 template<int D> struct Gauss_z2 {

```

```

3c1    bitset<D> basis[D], keep[D];
b16    int rk, in;
482    vector<int> id;

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

04e    bool add(bitset<D> v) {
42c        in++;
fb0        bitset<D> k;
659        for (int i = D - 1; i >= 0; i--) if (v[i]) {
189            if (basis[i][i]) v ^= basis[i], k ^= keep[i];
4e6            else {
ea6                k[i] = true, id[i] = in, keep[i] = k;
6ce                basis[i] = v, rk++;
8a6                return true;
cbb            }
cbb        }
dif        return false;
cbb    }

0f6    pair<bool, bitset<D>> coord(bitset<D> v) {
944        bitset<D> c;
659        for (int i = D - 1; i >= 0; i--) if (v[i]) {
a39            if (basis[i][i]) v ^= basis[i], c[i] = true;
8af            else return {false, bitset<D>()};
cbb        }
5db        return {true, c};
cbb    }

330    pair<bool, vector<int>> recover(bitset<D> v) {
22e        auto [span, bc] = coord(v);
af8        if (not span) return {false, {}};
f79        bitset<D> aux;
5a0        for (int i = D - 1; i >= 0; i--) if (bc[i]) aux
^= keep[i];
ea9        vector<int> oc;
ef2        for (int i = D - 1; i >= 0; i--) if (aux[i])
oc.push_back(id[i]);
001        return {true, oc};
cbb    }
214 };

```

## 4.11 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
//
// 0(log(min(a, b)))
// 2e8259

c5e    template<typename T> tuple<ll, T, T> ext_gcd(ll a, ll b)
    {
3bd        if (!a) return {b, 0, 1};
c4b        auto [g, x, y] = ext_gcd<T>(b%a, a);
c59        return {g, y - b/a*x, x};
cbb    }

// numero de solucoes de a*[lx, rx] + b*[ly, ry] = c
14c    template<typename T = ll> // usar __int128 se for ate
1e18
2a4    ll diophantine(ll a, ll b, ll c, ll lx, ll rx, ll ly, ll
ry) {
c80        if (lx > rx or ly > ry) return 0;
a98        if (a == 0 and b == 0) return c ? 0 :
(rx-lx+1)*(ry-ly+1);
8ce        auto [g, x, y] = ext_gcd<T>(abs(a), abs(b));
9c3        if (c % g != 0) return 0;
249        if (a == 0) return (rx-lx+1)*(ly <= c/b and c/b <=
ry);
4ce        if (b == 0) return (ry-ly+1)*(lx <= c/a and c/a <=
rx);
fb1        x *= a/abs(a) * c/g, y *= b/abs(b) * c/g, a /= g, b
/= g;

b20        auto shift = [&](T qt) { x += qt*b, y -= qt*a; };
efa        auto test = [&](T& k, ll mi, ll ma, ll coef, int t) {
866            shift((mi - k)*t / coef);
79d            if (k < mi) shift(coef > 0 ? t : -t);
74d            if (k > ma) return pair<T, T>(rx+2, rx+1);
41f            T x1 = x;
633            shift((ma - k)*t / coef);
c5b            if (k > ma) shift(coef > 0 ? -t : t);
4a9            return pair<T, T>(x1, x);

```

```

214     };

639     auto [l1, r1] = test(x, lx, rx, b, 1);
38e     auto [l2, r2] = test(y, ly, ry, a, -1);
c43     if (l2 > r2) swap(l2, r2);
50a     T l = max(l1, l2), r = min(r1, r2);
339     if (l > r) return 0;
42f     ll k = (r-l) / abs(b) + 1;
839     return k; // solucoes: x = l + [0, k)*|b|
cbb }

```

## 4.12 Exponenciacao rapida

//  $(x^y \bmod m)$  em  $O(\log(y))$

```

03c ll pow(ll x, ll y, ll m) { // iterativo
c85     ll ret = 1;
1b8     while (y) {
895         if (y & 1) ret = (ret * x) % m;
23b         y >>= 1;
cc5         x = (x * x) % m;
cbb     }
edf     return ret;
cbb }

```

```

03c ll pow(ll x, ll y, ll m) { // recursivo
13a     if (!y) return 1;
426     ll ans = pow(x*x%m, y/2, m);
88d     return y%2 ? x*ans%m : ans;
cbb }

```

## 4.13 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

```

```

382 template<char op, class T> vector<T> FWHT(vector<T> f,
bool inv = false) {
b75     int n = f.size();
d78     for (int k = 0; (n-1)>>k; k++) for (int i = 0; i <
n; i++) if (i>>k&1) {
29e         int j = i^(1<<k);
627         if (op == '^') f[j] += f[i], f[i] = f[j] -
2*f[i];
a38         if (op == '|') f[i] += (inv ? -1 : 1) * f[j];
93c         if (op == '&') f[j] += (inv ? -1 : 1) * f[i];
cbb     }
578     if (op == '^' and inv) for (auto& i : f) i /= n;
abe     return f;
cbb }

```

## 4.14 FFT

```

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

// Para FFT
// de56b9
488 void get_roots(bool f, int n, vector<complex<double>>&
roots) {
f26     const static double PI = acos(-1);
71a     for (int i = 0; i < n/2; i++) {
b1e         double alpha = i*((2*PI)/n);
1a1         if (f) alpha = -alpha;
069         roots[i] = {cos(alpha), sin(alpha)};
cbb     }
cbb }

// Para NTT
// 91cd08
9f7 template<int p>
97b void get_roots(bool f, int n, vector<mod_int<p>>& roots)
{
1e6     mod_int<p> r;
de9     int ord;
57a     if (p == 998244353) {

```

```

9b6         r = 102292;
81b         ord = (1 << 23);
1cc     } else if (p == 754974721) {
43a         r = 739831874;
f0a         ord = (1 << 24);
b60     } else if (p == 167772161) {
a2a         r = 243;
033         ord = (1 << 25);
6e0     } else assert(false);

547     if (f) r = r^(p - 1 -ord/n);
ee2     else r = r^(ord/n);
be4     roots[0] = 1;
078     for (int i = 1; i < n/2; i++) roots[i] =
        roots[i-1]*r;
cbb }

    // d5c432
8a2 template<typename T> void fft(vector<T> &a, bool f, int
N, vector<int> &rev) {
bc7     for (int i = 0; i < N; i++) if (i < rev[i])
        swap(a[i], a[rev[i]]);
12b     int l, r, m;
cb4     vector<T> roots(N);
192     for (int n = 2; n <= N; n *= 2) {
0f4         get_roots(f, n, roots);

5dc         for (int pos = 0; pos < N; pos += n) {
432             l = pos+0, r = pos+n/2, m = 0;
a88             while (m < n/2) {
297                 auto t = roots[m]*a[r];
254                 a[r] = a[l] - t;
b8f                 a[l] = a[l] + t;
925                 l++; r++; m++;
cbb             }
cbb         }
cbb     }
235     if (f) {
1c5         auto invN = T(1)/T(N);
557         for (int i = 0; i < N; i++) a[i] = a[i]*invN;
cbb     }
cbb }

```

```

bf5 template<typename T> vector<T> convolution(vector<T> &a,
vector<T> &b) {
279     vector<T> l(a.begin(), a.end());
f41     vector<T> r(b.begin(), b.end());
7c6     int ln = l.size(), rn = r.size();
287     int N = ln+rn-1;
f03     int n = 1, log_n = 0;
ac4     while (n <= N) { n <=<= 1; log_n++; }
808     vector<int> rev(n);
bae     for (int i = 0; i < n; ++i) {
434         rev[i] = 0;
920         for (int j = 0; j < log_n; ++j)
836             if (i & (1<<j)) rev[i] |= 1 << (log_n-1-j);
cbb     }
143     assert(N <= n);
fa4     l.resize(n);
7e4     r.resize(n);
56e     fft(l, false, n, rev);
fcf     fft(r, false, n, rev);
917     for (int i = 0; i < n; i++) l[i] *= r[i];
88b     fft(l, true, n, rev);
5e1     l.resize(N);
792     return l;
cbb }

    // NTT
    // 3bf256
6c8 template<int p, typename T> vector<mod_int<p>>
ntt(vector<T>& a, vector<T>& b) {
d52     vector<mod_int<p>> A(a.begin(), a.end()),
B(b.begin(), b.end());
d29     return convolution(A, B);
cbb }

    // Convolucao de inteiro
    //
    // Precisa do CRT
    //
    // Tabela de valores:
    // [0,1] - <int, 1>
    // [-1e5, 1e5] - <ll, 2>
    // [-1e9, 1e9] - <__int128, 3>

```

```

//
// 053a7d
b3c template<typename T, int mods>
eec vector<T> int_convolution(vector<int>& a, vector<int>&
b) {
fe8     static const int M1 = 998244353, M2 = 754974721, M3
= 167772161;

bf5     auto c1 = ntt<M1>(a, b);
221     auto c2 = (mods >= 2 ? ntt<M2>(a, b) :
vector<mod_int<M2>>());
f9b     auto c3 = (mods >= 3 ? ntt<M3>(a, b) :
vector<mod_int<M3>>());

2da     vector<T> ans;
5c5     for (int i = 0; i < c1.size(); i++) {
c09         crt<T> at(c1[i].v, M1);
316         if (mods >= 2) at = at * crt<T>(c2[i].v, M2);
987         if (mods >= 3) at = at * crt<T>(c3[i].v, M3);
b2b         ans.push_back(at.a);
26d         if (at.a > at.m/2) ans.back() -= at.m;
cbb     }
ba7     return ans;
cbb }

```

## 4.15 Integração Numérica - Método de Simpson 3/8

```

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

676 const int N = 3*100; // multiplo de 3
287 ld integrate(ld a, ld b, function<ld(ld)> f) {
b4d     ld s = 0, h = (b - a)/N;
067     for (int i = 1 ; i < N; i++) s += f(a + i*h)*(i%3 ?
3 : 2);
0da     return (f(a) + s + f(b))*3*h/8;
cbb }

```

## 4.16 Inverso Modular

```

// Computa o inverso de a modulo b

```

```

// Se b eh primo, basta fazer
// a^(b-2)

f0a ll inv(ll a, ll b) {
ae1     return a > 1 ? b - inv(b%a, a)*b/a : 1;
cbb }

// computa o inverso modular de 1..MAX-1 modulo um primo
a88 ll inv[MAX]:
0f2 inv[1] = 1;
0fa for (int i = 2; i < MAX; i++) inv[i] = MOD -
MOD/i*inv[MOD%i]%MOD;

```

## 4.17 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")
77a template<typename T> void kar(T* a, T* b, int n, T* r,
T* tmp) {
d4c     if (n <= 64) {
510         for (int i = 0; i < n; i++) for (int j = 0; j <
n; j++)
212             r[i+j] += a[i] * b[j];
505         return;
cbb     }
194     int mid = n/2;
2d7     T *atmp = tmp, *btmp = tmp+mid, *E = tmp+n;
4f1     memset(E, 0, sizeof(E[0])*n);
c65     for (int i = 0; i < mid; i++) {
c72         atmp[i] = a[i] + a[i+mid];
4b9         btmp[i] = b[i] + b[i+mid];
cbb     }
38a     kar(atmp, btmp, mid, E, tmp+2*n);
b1e     kar(a, b, mid, r, tmp+2*n);
229     kar(a+mid, b+mid, mid, r+n, tmp+2*n);
c65     for (int i = 0; i < mid; i++) {

```

```

735         T temp = r[i+mid];
de7         r[i+mid] += E[i] - r[i] - r[i+2*mid];
f1e         r[i+2*mid] += E[i+mid] - temp - r[i+3*mid];
cbb     }
cbb }

e38 template<typename T> vector<T> karatsuba(vector<T> a,
      vector<T> b) {
ba3     int n = max(a.size(), b.size());
a84     while (n&(n-1)) n++;
ca9     a.resize(n), b.resize(n);
ae0     vector<T> ret(2*n), tmp(4*n);
644     kar(&a[0], &b[0], n, &ret[0], &tmp[0]);
edf     return ret;
cbb }

```

## 4.18 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
//
// 0(sqrt(m) * log(sqrt(m)))
// 739fa8
d41
da8 int dlog(int b, int a, int m) {
9f8     if (a == 0) return b ? -1 : 1; // caso nao definido
d41
a6e     a %= m, b %= m;
a10     int k = 1, shift = 0;
31e     while (1) {
6e3         int g = gcd(a, m);
d47         if (g == 1) break;
d41
9bc         if (b == k) return shift;
642         if (b % g) return -1;
c36         b /= g, m /= g, shift++;
9ab         k = (ll) k * a / g % m;
cbb     }
d41
af7     int sq = sqrt(m)+1, giant = 1;

```

```

975     for (int i = 0; i < sq; i++) giant = (ll) giant * a
      % m;
d41
0b5     vector<pair<int, int>> baby;
33f     for (int i = 0, cur = b; i <= sq; i++) {
496         baby.emplace_back(cur, i);
16c         cur = (ll) cur * a % m;
cbb     }
eb4     sort(baby.begin(), baby.end());
d41
9c9     for (int j = 1, cur = k; j <= sq; j++) {
ace         cur = (ll) cur * giant % m;
78b         auto it = lower_bound(baby.begin(), baby.end(),
      pair(cur, INF));
d26         if (it != baby.begin() and (--it)->first == cur)
ac3             return sq * j - it->second + shift;
cbb     }
d41
daa     return -1;
cbb }

```

## 4.19 Miller-Rabin

```

// Testa se n eh primo, n <= 3 * 10^18
//
// 0(log(n)), considerando multiplicacao
// e exponenciacao constantes
// 4ebecc
d8b ll mul(ll a, ll b, ll m) {
e7a     ll ret = a*b - ll((long double)1/m*a*b+0.5)*m;
074     return ret < 0 ? ret+m : ret;
cbb }

03c ll pow(ll x, ll y, ll m) {
13a     if (!y) return 1;
dbc     ll ans = pow(mul(x, x, m), y/2, m);
7fa     return y%2 ? mul(x, ans, m) : ans;
cbb }

1a2 bool prime(ll n) {
1aa     if (n < 2) return 0;

```

```

237     if (n <= 3) return 1;
9de     if (n % 2 == 0) return 0;
f6a     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
771     for (int a : {2, 325, 9375, 28178, 450775, 9780504,
795265022}) {
da0         ll x = pow(a, d, n);
709         if (x == 1 or x == n - 1 or a % n == 0) continue;

4a2         for (int j = 0; j < r - 1; j++) {
10f             x = mul(x, x, n);
df0             if (x == n - 1) break;
cbb         }
e1b         if (x != n - 1) return 0;
cbb     }
6a5     return 1;
cbb }

```

## 4.20 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

d8b ll mul(ll a, ll b, ll m) {
e7a     ll ret = a*b - ll((long double)1/m*a*b+0.5)*m;
074     return ret < 0 ? ret+m : ret;
cbb }

03c ll pow(ll x, ll y, ll m) {
13a     if (!y) return 1;
dbc     ll ans = pow(mul(x, x, m), y/2, m);

```

```

7fa     return y%2 ? mul(x, ans, m) : ans;
cbb }

1a2 bool prime(ll n) {
1aa     if (n < 2) return 0;
237     if (n <= 3) return 1;
9de     if (n % 2 == 0) return 0;

f6a     ll r = __builtin_ctzll(n - 1), d = n >> r;
771     for (int a : {2, 325, 9375, 28178, 450775, 9780504,
795265022}) {
da0         ll x = pow(a, d, n);
709         if (x == 1 or x == n - 1 or a % n == 0) continue;

4a2         for (int j = 0; j < r - 1; j++) {
10f             x = mul(x, x, n);
df0             if (x == n - 1) break;
cbb         }
e1b         if (x != n - 1) return 0;
cbb     }
6a5     return 1;
cbb }

9cf ll rho(ll n) {
0f9     if (n == 1 or prime(n)) return n;
f7c     auto f = [n](ll x) {return mul(x, x, n) + 1;};

8a5     ll x = 0, y = 0, t = 30, prd = 2, x0 = 1, q;
533     while (t % 40 != 0 or gcd(prd, n) == 1) {
8a0         if (x==y) x = ++x0, y = f(x);
e13         q = mul(prd, abs(x-y), n);
21f         if (q != 0) prd = q;
450         x = f(x), y = f(f(y)), t++;
cbb     }
002     return gcd(prd, n);
cbb }

5b7 vector<ll> fact(ll n) {
1b9     if (n == 1) return {};
0ec     if (prime(n)) return {n};
0ed     ll d = rho(n);
1de     vector<ll> l = fact(d), r = fact(n / d);

```



```

3af      l.insert(l.end(), r.begin(), r.end());
792      return l;
cbb }

```

## 4.21 Produto de dois long long mod m

```

// 0(1)
// 260e72

d8b ll mul(ll a, ll b, ll m) { // a*b % m
e7a      ll ret = a*b - ll((long double)1/m*a*b+0.5)*m;
074      return ret < 0 ? ret+m : ret;
cbb }

```

## 4.22 Simplex

```

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

395 const double eps = 1e-7;

493 namespace Simplex {
69c     vector<vector<double>> T;
14e     int n, m;
43e     vector<int> X, Y;

c51     void pivot(int x, int y) {
8e6         swap(X[y], Y[x-1]);
d03         for (int i = 0; i <= m; i++) if (i != y) T[x][i]
/= T[x][y];
33c         T[x][y] = 1/T[x][y];
38b         for (int i = 0; i <= n; i++) if (i != x and
abs(T[i][y]) > eps) {
774             for (int j = 0; j <= m; j++) if (j != y)
T[i][j] -= T[i][y] * T[x][j];
3d8             T[i][y] = -T[i][y] * T[x][y];
cbb         }
cbb     }

```

```

// Retorna o par (valor maximo, vetor solucao)
6f8     pair<double, vector<double>> simplex(
e9d         vector<vector<double>> A, vector<double> b,
vector<double> c) {
5bb         n = b.size(), m = c.size();
002         T = vector(n + 1, vector<double>(m + 1));
2d9         X = vector<int>(m);
0c2         Y = vector<int>(n);
115         for (int i = 0; i < m; i++) X[i] = i;
51f         for (int i = 0; i < n; i++) Y[i] = i+m;
5b5         for (int i = 0; i < m; i++) T[0][i] = -c[i];
603         for (int i = 0; i < n; i++) {
ba6             for (int j = 0; j < m; j++) T[i+1][j] =
A[i][j];
eca             T[i+1][m] = b[i];
cbb         }
667         while (true) {
714             int x = -1, y = -1;
2db             double mn = -eps;
c29             for (int i = 1; i <= n; i++) if (T[i][m] <
mn) mn = T[i][m], x = i;
af2             if (x < 0) break;
882             for (int i = 0; i < m; i++) if (T[x][i] <
-eps) { y = i; break; }

4a6             if (y < 0) return {-1e18, {}}; // sem
solucao para Ax <= b
7fb             pivot(x, y);
cbb         }
667         while (true) {
714             int x = -1, y = -1;
2db             double mn = -eps;
562             for (int i = 0; i < m; i++) if (T[0][i] <
mn) mn = T[0][i], y = i;
9b0             if (y < 0) break;
034             mn = 1e200;
5af             for (int i = 1; i <= n; i++) if (T[i][y] >
eps and T[i][m] / T[i][y] < mn)
48f                 mn = T[i][m] / T[i][y], x = i;

53b             if (x < 0) return {1e18, {}}; // c^T x eh
ilimitado

```

```

7fb          pivot(x, y);
cbb      }
290          vector<double> r(m);
32f          for(int i = 0; i < n; i++) if (Y[i] < m) r[Y[i]]
= T[i+1][m];
e59          return {T[0][m], r};
cbb      }
cbb }

```

## 4.23 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

```

```

153 template<typename T> tuple<T, T, T> ext_gcd(T a, T b) {
3bd     if (!a) return {b, 0, 1};
550     auto [g, x, y] = ext_gcd(b%a, a);
c59     return {g, y - b/a*x, x};
cbb }

```

```

bfe template<typename T = ll> struct crt {
627     T a, m;

```

```

5f3     crt() : a(0), m(1) {}
7eb     crt(T a_, T m_) : a(a_), m(m_) {}
911     crt operator * (crt C) {
238         auto [g, x, y] = ext_gcd(m, C.m);
dc0         if ((a - C.a) % g) a = -1;
4f9         if (a == -1 or C.a == -1) return crt(-1, 0);
d09         T lcm = m/g*C.m;
eb2         T ans = a + (x*(C.a-a)/g % (C.m/g))*m;
d8d         return crt((ans % lcm + lcm) % lcm, lcm);
cbb     }
214 };

```

## 4.24 Totiente

```

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

```

```

a7e int tot(int n){
0f6     int ret = n;

505     for (int i = 2; i*i <= n; i++) if (n % i == 0) {
b0c         while (n % i == 0) n /= i;
125         ret -= ret / i;
cbb     }
af4     if (n > 1) ret -= ret / n;

edf     return ret;
cbb }

```

## 4.25 Variacoes do 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))

```

```

f12 int divi[MAX];

fb9 void crivo(int lim) {
f53     for (int i = 1; i <= lim; i++) divi[i] = 1;

d46     for (int i = 2; i <= lim; i++) if (divi[i] == 1)
018         for (int j = i; j <= lim; j += i) divi[j] = i;
cbb }

470 void fact(vector<int>& v, int n) {
ac8     if (n != divi[n]) fact(v, n/divi[n]);
ab4     v.push_back(divi[n]);
cbb }

```

```

// Crivo linear

```

```

//
// Mesma coisa que o de cima, mas tambem
// calcula a lista de primos
//
// O(n)

f12 int divi[MAX];
fd3 vector<int> primes;

fb9 void crivo(int lim) {
d5a     divi[1] = 1;
f70     for (int i = 2; i <= lim; i++) {
3eb         if (divi[i] == 0) divi[i] = i,
primes.push_back(i);
3ba         for (int j : primes) {
522             if (j > divi[i] or i*j > lim) break;
00b             divi[i*j] = j;
cbb         }
cbb     }
cbb }

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

f12 int divi[MAX];

fb9 void crivo(int lim) {
f53     for (int i = 1; i <= lim; i++) divi[i] = 1;

424     for (int i = 2; i <= lim; i++)
594         for (int j = i; j <= lim; j += i) {
// para numero de divisores
divi[j]++;
// para soma dos divisores
278         divi[j] += i;
cbb     }
cbb }

```

```

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

5f4 int tot[MAX];

fb9 void crivo(int lim) {
a27     for (int i = 1; i <= lim; i++) {
bc9         tot[i] += i;
feb         for (int j = 2*i; j <= lim; j += i)
837             tot[j] -= tot[i];
cbb     }
cbb }

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

4e1 char meb[MAX];

fb9 void crivo(int lim) {
649     for (int i = 2; i <= lim; i++) meb[i] = 2;
ace     meb[1] = 1;
842     for (int i = 2; i <= lim; i++) if (meb[i] == 2)
8d8         for (int j = i; j <= lim; j += i) if (meb[j]) {
686             if (meb[j] == 2) meb[j] = 1;
ae1             meb[j] *= j/i%i ? -1 : 0;
cbb         }
cbb }

// 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

```

```

//
// 0(n)

fd3 vector<int> primes;
623 int f[MAX], pot[MAX];
    //int expo[MAX];

5c4 void sieve(int lim) {
    // Funcoes para soma dos divisores:
fc9     auto f_prime = [](int p) { return p+1; };
31c     auto add_prime = [](int fpak, int p) { return
        fpak*p+1; };
        //auto f_pak = [](int p, int k) {};

02d     f[1] = 1;
f70     for (int i = 2; i <= lim; i++) {
e6b         if (!pot[i]) {
e74             primes.push_back(i);
f05             f[i] = f_prime(i), pot[i] = i;
                //expo[i] = 1;

cbb         }
3b9         for (int p : primes) {
b9f             if (i*p > lim) break;
569             if (i%p == 0) {
b97                 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;
51f                 pot[i*p] = pot[i] * p;
c2b                 break;
9d9             } else {
9ef                 f[i*p] = f[i] * f[p];
638                 pot[i*p] = p;
                    //expo[i*p] = 1;

cbb             }
cbb         }
cbb     }
cbb }

```

## 5 DP

### 5.1 Convex Hull Trick (Rafael)

```

// adds tem que serem feitos em ordem de slope
// queries tem que ser feitas em ordem de x
//
// linear
// 30323e

4b5 struct CHT {
942     int it;
ac1     vector<ll> a, b;
45e     CHT():it(0){}
0bb     ll eval(int i, ll x){
93d         return a[i]*x + b[i];
cbb     }
63a     bool useless(){
a20         int sz = a.size();
35f         int r = sz-1, m = sz-2, l = sz-3;
d71         return (b[l] - b[r])*(a[m] - a[l]) <
413             (b[l] - b[m])*(a[r] - a[l]);
cbb     }
bf4     void add(ll A, ll B){
7f5         a.push_back(A); b.push_back(B);
565         while (!a.empty()){
233             if ((a.size() < 3) || !useless()) break;
ecb             a.erase(a.end() - 2);
568             b.erase(b.end() - 2);
cbb         }
cbb     }
81b     ll get(ll x){
d27         it = min(it, int(a.size()) - 1);
46a         while (it+1 < a.size()){
3c4             if (eval(it+1, x) > eval(it, x)) it++;
f97             else break;
cbb         }
420         return eval(it, x);
cbb     }
214 };

```

## 5.2 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
//
// 0(log(n)) amortizado por insercao
// 0(log(n)) por query
// 978376

72c struct Line {
073     mutable ll a, b, p;
8e3     bool operator<(const Line& o) const { return a <
        o.a; }
abf     bool operator<(ll x) const { return p < x; }
214 };

326 struct dynamic_hull : multiset<Line, less<>> {
33a     ll div(ll a, ll b) {
a20         return a / b - ((a ^ b) < 0 and a % b);
cbb     }

bbb     void update(iterator x) {
b2a         if (next(x) == end()) x->p = LINF;
772         else if (x->a == next(x)->a) x->p = x->b >=
            next(x)->b ? LINF : -LINF;
424         else x->p = div(next(x)->b - x->b, x->a -
            next(x)->a);
cbb     }

71c     bool overlap(iterator x) {
f18         update(x);
cfa         if (next(x) == end()) return 0;
a4a         if (x->a == next(x)->a) return x->b >=
            next(x)->b;
d40         return x->p >= next(x)->p;
cbb     }

176     void add(ll a, ll b) {
1c7         auto x = insert({a, b, 0});
```

```
4ab         while (overlap(x)) erase(next(x)), update(x);
dbc         if (x != begin() and !overlap(prev(x))) x =
            prev(x), update(x);
0fc         while (x != begin() and overlap(prev(x)))
4d2             x = prev(x), erase(next(x)), update(x);
cbb     }

4ad     ll query(ll x) {
229         assert(!empty());
7d1         auto l = *lower_bound(x);
aba         return l.a * x + l.b;
cbb     }
214 };
```

## 5.3 Divide and Conquer DP

```
// Particiona o array em k subarrays
// minimizando o somatorio das queries
//
// 0(k n log n), assumindo quer query(l, r) eh 0(1)
// 4efe6b

547 ll dp[MAX][2];

94b void solve(int k, int l, int r, int lk, int rk) {
de6     if (l > r) return;
109     int m = (l+r)/2, p = -1;
d2b     auto& ans = dp[m][k&1] = LINF;
6e2     for (int i = max(m, lk); i <= rk; i++) {
324         int at = dp[i+1][~k&1] + query(m, i);
57d         if (at < ans) ans = at, p = i;
cbb     }
1ee     solve(k, l, m-1, lk, p), solve(k, m+1, r, p, rk);
cbb }

cf1 ll DC(int n, int k) {
321     dp[n][0] = dp[n][1] = 0;
f27     for (int i = 0; i < n; i++) dp[i][0] = LINF;
b76     for (int i = 1; i <= k; i++) solve(i, 0, n-i, 0,
        n-i);
8e7     return dp[0][k&1];
cbb }
```

## 5.4 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

eaf int lcs_s[MAX], lcs_t[MAX];
a6d int dp[2][MAX];

// dp[0][j] = max lcs(s[li...ri], t[lj, lj+j])
d12 void dp_top(int li, int ri, int lj, int rj) {
d13     memset(dp[0], 0, (rj-lj+1)*sizeof(dp[0][0]));
753     for (int i = li; i <= ri; i++) {
9aa         for (int j = rj; j >= lj; j--)
83b             dp[0][j - lj] = max(dp[0][j - lj],
741             (lcs_s[i] == lcs_t[j]) + (j > lj ? dp[0][j-1
- lj] : 0));
04c         for (int j = lj+1; j <= rj; j++)
939             dp[0][j - lj] = max(dp[0][j - lj], dp[0][j-1
- lj]);
cbb     }
cbb }

// dp[1][j] = max lcs(s[li...ri], t[lj+j, rj])
ca0 void dp_bottom(int li, int ri, int lj, int rj) {
0dd     memset(dp[1], 0, (rj-lj+1)*sizeof(dp[1][0]));
3a2     for (int i = ri; i >= li; i--) {
49c         for (int j = lj; j <= rj; j++)
dbb             dp[1][j - lj] = max(dp[1][j - lj],
4da             (lcs_s[i] == lcs_t[j]) + (j < rj ?
dp[1][j+1 - lj] : 0));
6ca         for (int j = rj-1; j >= lj; j--)
769             dp[1][j - lj] = max(dp[1][j - lj], dp[1][j+1
- lj]);
cbb     }
cbb }

93c void solve(vector<int>& ans, int li, int ri, int lj, int
rj) {
2ad     if (li == ri){
```

```
49c         for (int j = lj; j <= rj; j++)
f5b             if (lcs_s[li] == lcs_t[j]){
a66                 ans.push_back(lcs_t[j]);
c2b                 break;
cbb             }
505         return;
cbb     }
534     if (lj == rj){
753         for (int i = li; i <= ri; i++){
88f             if (lcs_s[i] == lcs_t[lj]){
531                 ans.push_back(lcs_s[i]);
c2b                 break;
cbb             }
cbb         }
505         return;
cbb     }
a57     int mi = (li+ri)/2;
ade     dp_top(li, mi, lj, rj), dp_bottom(mi+1, ri, lj, rj);

d7a     int j_ = 0, mx = -1;

aee     for (int j = lj-1; j <= rj; j++) {
da8         int val = 0;
2bb         if (j >= lj) val += dp[0][j - lj];
b9e         if (j < rj) val += dp[1][j+1 - lj];

ba8         if (val >= mx) mx = val, j_ = j;
cbb     }
6f1     if (mx == -1) return;
c2a     solve(ans, li, mi, lj, j_), solve(ans, mi+1, ri,
j_+1, rj);
cbb }

058 vector<int> lcs(const vector<int>& s, const vector<int>&
t) {
953     for (int i = 0; i < s.size(); i++) lcs_s[i] = s[i];
577     for (int i = 0; i < t.size(); i++) lcs_t[i] = t[i];
dab     vector<int> ans;
599     solve(ans, 0, s.size()-1, 0, t.size()-1);
ba7     return ans;
cbb }
```

## 5.5 Mochila

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

add int v[MAX], w[MAX]; // valor e peso
582 int dp[2][MAX_CAP];

// DP usando os itens [l, r], com capacidade = cap
0d6 void get_dp(int x, int l, int r, int cap) {
f8f     memset(dp[x], 0, (cap+1)*sizeof(dp[x][0]));
574     for (int i = l; i <= r; i++) for (int j = cap; j >=
0; j--)
3a9         if (j - w[i] >= 0) dp[x][j] = max(dp[x][j], v[i]
+ dp[x][j - w[i]]);
cbb }

5ab void solve(vector<int>& ans, int l, int r, int cap) {
893     if (l == r) {
9ff         if (w[l] <= cap) ans.push_back(l);
505         return;
cbb     }
ee4     int m = (l+r)/2;
283     get_dp(0, l, m, cap), get_dp(1, m+1, r, cap);
056     int left_cap = -1, opt = -INF;
c94     for (int j = 0; j <= cap; j++)
2f2         if (int at = dp[0][j] + dp[1][cap - j]; at > opt)
91d             opt = at, left_cap = j;
da3     solve(ans, l, m, left_cap), solve(ans, m+1, r, cap -
left_cap);
cbb }

0d7 vector<int> knapsack(int n, int cap) {
dab     vector<int> ans;
1e0     solve(ans, 0, n-1, cap);
ba7     return ans;
cbb }
```

## 5.6 SOS DP

```
// O(n 2^n)

// soma de sub-conjunto
e03 vector<ll> sos_dp(vector<ll> f) {
6c0     int N = __builtin_ctz(f.size());
e59     assert((1<<N) == f.size());

5a5     for (int i = 0; i < N; i++) for (int mask = 0; mask
< (1<<N); mask++)
796         if (mask>>i&1) f[mask] += f[mask^(1<<i)];
abe     return f;
cbb }

// soma de super-conjunto
e03 vector<ll> sos_dp(vector<ll> f) {
6c0     int N = __builtin_ctz(f.size());
e59     assert((1<<N) == f.size());

5a5     for (int i = 0; i < N; i++) for (int mask = 0; mask
< (1<<N); mask++)
a3c         if (~mask>>i&1) f[mask] += f[mask^(1<<i)];
abe     return f;
cbb }

// 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

ea1 namespace aho {
807     map<char, int> to[MAX];
c87     int link[MAX], idx, term[MAX], exit[MAX], sobe[MAX];
```

```

bfc void insert(string& s) {
05e int at = 0;
b4f for (char c : s) {
b68 auto it = to[at].find(c);
1c9 if (it == to[at].end()) at = to[at][c] =
++idx;
361 else at = it->second;
cbb }
142 term[at]++, sobe[at]++;
cbb }
d41 #warning nao esquece de chamar build() depois de inserir
0a8 void build() {
26a queue<int> q;
537 q.push(0);
dff link[0] = exit[0] = -1;
402 while (q.size()) {
379 int i = q.front(); q.pop();
3c4 for (auto [c, j] : to[i]) {
5da int l = link[i];
102 while (l != -1 and !to[l].count(c)) l =
link[l];
7a5 link[j] = l == -1 ? 0 : to[l][c];
3ab exit[j] = term[link[j]] ? link[j] :
exit[link[j]];
6f2 if (exit[j]+1) sobe[j] += sobe[exit[j]];
113 q.push(j);
cbb }
cbb }
cbb }
bc0 int query(string& s) {
86d int at = 0, ans = 0;
b4f for (char c : s){
1ca while (at != -1 and !to[at].count(c)) at =
link[at];
5b9 at = at == -1 ? 0 : to[at][c];
2b1 ans += sobe[at];
cbb }
ba7 return ans;
cbb }
cbb }

```

## 6.2 Algoritmo Z

```

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

a19 vector<int> get_z(string s) {
163 int n = s.size();
2b1 vector<int> z(n, 0);

fae int l = 0, r = 0;
6f5 for (int i = 1; i < n; i++) {
0af if (i <= r) z[i] = min(r - i + 1, z[i - 1]);
457 while (i + z[i] < n and s[z[i]] == s[i + z[i]])
z[i]++;
65e if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
cbb }

070 return z;
cbb }

```

## 6.3 Automato de Sufixo

```

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

16e namespace sam {
c1a int cur, sz, len[2*MAX], link[2*MAX], acc[2*MAX];
0b8 int nxt[2*MAX][26];

e6a void add(int c) {
17a int at = cur;
9a6 len[sz] = len[cur]+1, cur = sz++;
500 while (at != -1 and !nxt[at][c]) nxt[at][c] =
cur, at = link[at];
7ea if (at == -1) { link[cur] = 0; return; }
654 int q = nxt[at][c];

```



```

fd9         if (len[q] == len[at]+1) { link[cur] = q;
return; }
31f         int qq = sz++;
2c3         len[qq] = len[at]+1, link[qq] = link[q];
9a9         for (int i = 0; i < 26; i++) nxt[qq][i] =
nxt[q][i];
e76         while (at != -1 and nxt[at][c] == q) nxt[at][c]
= qq, at = link[at];
8b8         link[cur] = link[q] = qq;
cbb     }
94e     void build(string& s) {
889         cur = 0, sz = 0, len[0] = 0, link[0] = -1, sz++;
9fe         for (auto i : s) add(i-'a');
17a         int at = cur;
121         while (at) acc[at] = 1, at = link[at];
cbb     }

// coisas que da pra fazer:
28c     ll distinct_substrings() {
04b         ll ans = 0;
a1e         for (int i = 1; i < sz; i++) ans += len[i] -
len[link[i]];
ba7         return ans;
cbb     }
a6c     string longest_common_substring(string& S, string&
T) {
419         build(S);
111         int at = 0, l = 0, ans = 0, pos = -1;
d59         for (int i = 0; i < T.size(); i++) {
f2c             while (at and !nxt[at][T[i]-'a']) at =
link[at], l = len[at];
efa             if (nxt[at][T[i]-'a']) at =
nxt[at][T[i]-'a'], l++;
749             else at = 0, l = 0;
a1a             if (l > ans) ans = l, pos = i;
cbb         }
20f         return T.substr(pos-ans+1, ans);
cbb     }
46e     ll dp[2*MAX];
455     ll paths(int i) {
2a8         auto& x = dp[i];
dee         if (x) return x;

```

```

483         x = 1;
71c         for (int j = 0; j < 26; j++) if (nxt[i][j]) x +=
paths(nxt[i][j]);
ea5         return x;
cbb     }
105     void kth_substring(int k, int at=0) { // k=1 : menor
substring lexicog.
9d2         for (int i = 0; i < 26; i++) if (k and
nxt[at][i]) {
d58             if (paths(nxt[at][i]) >= k) {
d02                 cout << char('a'+i);
c43                 kth_substring(k-1, nxt[at][i]);
505                 return;
cbb             }
5f4             k -= paths(nxt[at][i]);
cbb         }
cbb     }
214 };

```

## 6.4 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. 0 propagate() retorna o numero de
// substrings pal. com repeticao
//
// 0(n) amortizado, considerando alfabeto 0(1)
// a2e693

8eb struct eertree {
7cc     vector<vector<int>> t;
42e     int n, last, sz;
745     vector<int> s, len, link, qt;

d36     eertree(int N) {
ec8         t = vector(N+2, vector(26, int()));
cee         s = len = link = qt = vector<int>(N+2);
cd1         s[0] = -1;
288         link[0] = 1, len[0] = 0, link[1] = 1, len[1] =
-1;

```

```

688         sz = 2, last = 0, n = 1;
cbb     }

244     void add(char c) {
692         s[n++] = c -= 'a';
34f         while (s[n-len[last]-2] != c) last = link[last];
289         if (!t[last][c]) {
dab             int prev = link[last];
553             while (s[n-len[prev]-2] != c) prev =
link[prev];
fb2             link[sz] = t[prev][c];
3f5             len[sz] = len[last]+2;
1f8             t[last][c] = sz++;
cbb         }
344         qt[last = t[last][c]]++;
cbb     }
f17     int size() { return sz-2; }
2af     ll propagate() {
b73         ll ret = 0;
ebb         for (int i = n; i > 1; i--) {
fd3             qt[link[i]] += qt[i];
db5             ret += qt[i];
cbb         }
edf         return ret;
cbb     }
214 };

```

## 6.5 KMP

```

// mathcing(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
ea8 template<typename T> vector<int> pi(T s) {
019     vector<int> p(s.size());

```

```

725     for (int i = 1, j = 0; i < s.size(); i++) {
a51         while (j and s[j] != s[i]) j = p[j-1];
973         if (s[j] == s[i]) j++;
f8c         p[i] = j;
cbb     }
74e     return p;
cbb }

// c82524
c10 template<typename T> vector<int> matching(T& s, T& t) {
658     vector<int> p = pi(s), match;
a1b     for (int i = 0, j = 0; i < t.size(); i++) {
6be         while (j and s[j] != t[i]) j = p[j-1];
c4d         if (s[j] == t[i]) j++;
310         if (j == s.size()) match.push_back(i-j+1), j =
p[j-1];
cbb     }
ed8     return match;
cbb }

// 79bd9e
a2d struct KMPaut : vector<vector<int>> {
47c     KMPaut(){}
6c7     KMPaut (string& s) : vector<vector<int>>(26,
vector<int>(s.size()+1)) {
503         vector<int> p = pi(s);
04b         auto& aut = *this;
4fa         aut[s[0]-'a'][0] = 1;
19a         for (char c = 0; c < 26; c++)
5d3             for (int i = 1; i <= s.size(); i++)
42b                 aut[c][i] = s[i]-'a' == c ? i+1 :
aut[c][p[i-1]];
cbb     }
214 };

```

## 6.6 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
28a template<typename T> vector<int> manacher(const T& s) {
18f     int l = 0, r = -1, n = s.size();
fc9     vector<int> d1(n), d2(n);
603     for (int i = 0; i < n; i++) {
821         int k = i > r ? 1 : min(d1[l+r-i], r-i);
61a         while (i+k < n && i-k >= 0 && s[i+k] == s[i-k])
            k++;
61e         d1[i] = k--;
9f6         if (i+k > r) l = i-k, r = i+k;
cbb     }
e03     l = 0, r = -1;
603     for (int i = 0; i < n; i++) {
a64         int k = i > r ? 0 : min(d2[l+r-i+1], r-i+1); k++;
2c6         while (i+k <= n && i-k >= 0 && s[i+k-1] ==
            s[i-k]) k++;
eaa         d2[i] = --k;
26d         if (i+k-1 > r) l = i-k, r = i+k-1;
cbb     }
c41     vector<int> ret(2*n-1);
e6b     for (int i = 0; i < n; i++) ret[2*i] = 2*d1[i]-1;
e1d     for (int i = 0; i < n-1; i++) ret[2*i+1] = 2*d2[i+1];
edf     return ret;
cbb }

// 60c6f5
// verifica se a string s[i..j] eh palindromo
cac template<typename T> struct palindrome {
f97     vector<int> man;

b2d     palindrome(const T& s) : man(manacher(s)) {}
9d7     bool query(int i, int j) {
bad         return man[i+j] >= j-i+1;
cbb     }
214 };

// 8bd4d5

```

```

// tamanho do maior palindromo que termina em cada
// posicao
7cb template<typename T> vector<int> pal_end(const T& s) {
e57     vector<int> ret(s.size());
fde     palindrome<T> p(s);
d51     ret[0] = 1;
88e     for (int i = 1; i < s.size(); i++) {
a32         ret[i] = min(ret[i-1]+2, i+1);
6ea         while (!p.query(i-ret[i]+1, i)) ret[i]--;
cbb     }
edf     return ret;
cbb }

```

## 6.7 Min/max suffix/cyclic shift

```

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

016 template<typename T> int max_suffix(T s, bool mi =
    false) {
476     s.push_back(*min_element(s.begin(), s.end())-1);
1a4     int ans = 0;
88e     for (int i = 1; i < s.size(); i++) {
eec         int j = 0;
708         while (ans+j < i and s[i+j] == s[ans+j]) j++;
7a2         if (s[i+j] > s[ans+j]) {
b52             if (!mi or i != s.size()-2) ans = i;
c05         } else if (j) i += j-1;
cbb     }
ba7     return ans;
cbb }

a1a template<typename T> int min_suffix(T s) {
76b     for (auto& i : s) i *= -1;
09d     s.push_back(*max_element(s.begin(), s.end())+1);
925     return max_suffix(s, true);
cbb }

97c template<typename T> int max_cyclic_shift(T s) {

```

```

163     int n = s.size();
1ad     for (int i = 0; i < n; i++) s.push_back(s[i]);
20a     return max_suffix(s);
cbb }

08a template<typename T> int min_cyclic_shift(T s) {
76b     for (auto& i : s) i *= -1;
7be     return max_cyclic_shift(s);
cbb }

```

## 6.8 String Hashing

```

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

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

463 int uniform(int l, int r) {
a7f     uniform_int_distribution<int> uid(l, r);
f54     return uid(rng);
cbb }

9e0 template<int MOD> struct str_hash { // 116fcb
c63     static int P;
dcf     vector<ll> h, p;
ea8     str_hash(string s) : h(s.size()), p(s.size()) {
7a2         p[0] = 1, h[0] = s[0];
ad7         for (int i = 1; i < s.size(); i++)
84c             p[i] = p[i - 1]*P%MOD, h[i] = (h[i - 1]*P +
s[i])%MOD;
cbb     }
af7     ll operator()(int l, int r) { // retorna hash
s[l...r]
749         ll hash = h[r] - (l ? h[l - 1]*p[r - l + 1]%MOD
: 0);
dfd         return hash < 0 ? hash + MOD : hash;
cbb     }
214 };
217 template<int MOD> int str_hash<MOD>::P = uniform(256,
MOD - 1); // l > |sigma|

```

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

```

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

9d0 const ll MOD = (1ll<<61) - 1;
e38 ll mulmod(ll a, ll b) {
ff3     const static ll LOWER = (1ll<<30) - 1, GET31 =
(1ll<<31) - 1;
410     ll l1 = a&LOWER, h1 = a>>30, l2 = b&LOWER, h2 =
b>>30;
d54     ll m = l1*h2 + l2*h1, h = h1*h2;
784     ll ans = l1*l2 + (h>>1) + ((h&1)<<60) + (m>>31) +
((m&GET31)<<30) + 1;
1dd     ans = (ans&MOD) + (ans>>61), ans = (ans&MOD) +
(ans>>61);
c0f     return ans - 1;
cbb }

798 mt19937_64
    rng(chrono::steady_clock::now().time_since_epoch().count());

f89 ll uniform(ll l, ll r) {
969     uniform_int_distribution<ll> uid(l, r);
f54     return uid(rng);
cbb }

d7d struct str_hash {
c20     static ll P;
dcf     vector<ll> h, p;
ea8     str_hash(string s) : h(s.size()), p(s.size()) {
7a2         p[0] = 1, h[0] = s[0];
ad7         for (int i = 1; i < s.size(); i++)
632             p[i] = mulmod(p[i - 1], P), h[i] =
(mulmod(h[i - 1], P) + s[i])%MOD;
cbb     }
af7     ll operator()(int l, int r) { // retorna hash

```

```

s[1...r]
538     ll hash = h[r] - (l ? mulmod(h[l - 1], p[r - 1 +
    1]) : 0);
dfd     return hash < 0 ? hash + MOD : hash;
cbb     }
214 };
6c5 ll str_hash::P = uniform(256, MOD - 1); // l > |sigma|

```

## 6.10 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

733 vector<int> suffix_array(string s) {
b38     s += "$";
043     int n = s.size(), N = max(n, 260);
2f3     vector<int> sa(n), ra(n);
29b     for(int i = 0; i < n; i++) sa[i] = i, ra[i] = s[i];

0a2     for(int k = 0; k < n; k ? k *= 2 : k++) {
5ce         vector<int> nsa(sa), nra(n), cnt(N);

fae         for(int i = 0; i < n; i++) nsa[i] =
            (nsa[i]-k+n)%n, cnt[ra[i]]++;
4c4         for(int i = 1; i < N; i++) cnt[i] += cnt[i-1];
368         for(int i = n-1; i+1; i--) sa[--cnt[ra[nsa[i]]]]
            = nsa[i];

28f         for(int i = 1, r = 0; i < n; i++) nra[sa[i]] = r
            += ra[sa[i]] !=
f86             ra[sa[i-1]] or ra[(sa[i]+k)%n] !=
            ra[(sa[i-1]+k)%n];
26b         ra = nra;
d5e         if (ra[sa[n-1]] == n-1) break;
cbb     }
057     return vector<int>(sa.begin()+1, sa.end());
cbb }

```

```

481 vector<int> kasai(string s, vector<int> sa) {
232     int n = s.size(), k = 0;
408     vector<int> ra(n), lcp(n);
676     for (int i = 0; i < n; i++) ra[sa[i]] = i;

740     for (int i = 0; i < n; i++, k -= !!k) {
199         if (ra[i] == n-1) { k = 0; continue; }
1de         int j = sa[ra[i]+1];
891         while (i+k < n and j+k < n and s[i+k] == s[j+k])
            k++;
d98         lcp[ra[i]] = k;
cbb     }
5ed     return lcp;
cbb }

```

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

```

// Rapido
// 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 -  $O(1)$ 

// bab412
1a5 template<typename T> struct rmq {
517     vector<T> v;
fcc     int n; static const int b = 30;
70e     vector<int> mask, t;

183     int op(int x, int y) { return v[x] <= v[y] ? x : y; }
ee1     int msb(int x) { return
        __builtin_clz(1)-__builtin_clz(x); }
c92     int small(int r, int sz = b) { return
        r-msb(mask[r]&((1<sz)-1)); }
6ad     rmq() {}
43c     rmq(const vector<T>& v_) : v(v_), n(v.size()),
        mask(n), t(n) {
2e5         for (int i = 0, at = 0; i < n; mask[i++] = at |=

```

```

1) {
a61         at = (at<<1)&((1<<b)-1);
c00         while (at and op(i-msb(at&-at), i) == i) at
           ^= at&-at;
cbb     }
ea4     for (int i = 0; i < n/b; i++) t[i] =
        small(b*i+b-1);
39d     for (int j = 1; (1<<j) <= n/b; j++) for (int i =
        0; i+(1<<j) <= n/b; i++)
ba5         t[n/b*j+i] = op(t[n/b*(j-1)+i],
        t[n/b*(j-1)+i+(1<<(j-1))]);
cbb     }
e34     int index_query(int l, int r) {
27b         if (r-l+1 <= b) return small(r, r-l+1);
e80         int x = l/b+1, y = r/b-1;
fd3         if (x > y) return op(small(l+b-1), small(r));
a4e         int j = msb(y-x+1);
ea3         int ans = op(small(l+b-1), op(t[n/b*j+x],
        t[n/b*j+y-(1<<j)+1]));
be6         return op(ans, small(r));
cbb     }
093     T query(int l, int r) { return v[index_query(l, r)];
    }
214 };

9d7 struct suffix_array {
ac0     string s;
1a8     int n;
5b4     vector<int> sa, cnt, rnk, lcp;
2de     rmq<int> RMQ;

d6e     bool cmp(int a1, int b1, int a2, int b2, int a3=0,
        int b3=0) {
91d         return a1 != b1 ? a1 < b1 : (a2 != b2 ? a2 < b2
        : a3 < b3);
cbb     }
4a4     template<typename T> void radix(int* fr, int* to, T*
        r, int N, int k) {
c17         cnt = vector<int>(k+1, 0);
bac         for (int i = 0; i < N; i++) cnt[r[fr[i]]]++;
703         for (int i = 1; i <= k; i++) cnt[i] += cnt[i-1];
000         for (int i = N-1; i+1; i--) to[--cnt[r[fr[i]]]]

```

```

        = fr[i];
cbb     }
d66     void rec(vector<int>& v, int k) {
a76         auto &tmp = rnk, &m0 = lcp;
3a9         int N = v.size()-3, sz = (N+2)/3, sz2 = sz+N/3;
7f8         vector<int> R(sz2+3);
74f         for (int i = 1, j = 0; j < sz2; i += i%3) R[j++]
        = i;

b30         radix(&R[0], &tmp[0], &v[0]+2, sz2, k);
207         radix(&tmp[0], &R[0], &v[0]+1, sz2, k);
5f1         radix(&R[0], &tmp[0], &v[0]+0, sz2, k);

af5         int dif = 0;
ed9         int l0 = -1, l1 = -1, l2 = -1;
d81         for (int i = 0; i < sz2; i++) {
8de             if (v[tmp[i]] != l0 or v[tmp[i]+1] != l1 or
        v[tmp[i]+2] != l2)
b43                 l0 = v[tmp[i]], l1 = v[tmp[i]+1], l2 =
        v[tmp[i]+2], dif++;
199             if (tmp[i]%3 == 1) R[tmp[i]/3] = dif;
1f5             else R[tmp[i]/3+sz] = dif;
cbb         }

47f         if (dif < sz2) {
146             rec(R, dif);
746             for (int i = 0; i < sz2; i++) R[sa[i]] = i+1;
8b7             } else for (int i = 0; i < sz2; i++) sa[R[i]-1]
        = i;

6f4         for (int i = 0, j = 0; j < sz2; i++) if (sa[i] <
        sz) tmp[j++] = 3*sa[i];
7ce         radix(&tmp[0], &m0[0], &v[0], sz, k);
74d         for (int i = 0; i < sz2; i++)
c9e             sa[i] = sa[i] < sz ? 3*sa[i]+1 :
        3*(sa[i]-sz)+2;

332         int at = sz2+sz-1, p = sz-1, p2 = sz2-1;
1c9         while (p >= 0 and p2 >= 0) {
3b3             if ((sa[p2]%3==1 and cmp(v[m0[p]],
        v[sa[p2]], R[m0[p]/3],
0ce                 R[sa[p2]/3+sz])) or (sa[p2]%3==2 and

```

```

    cmp(v[m0[p]], v[sa[p2]],
af6      v[m0[p]+1], v[sa[p2]+1], R[m0[p]/3+sz],
      R[sa[p2]/3+1]))))
300      sa[at--] = sa[p2--];
cb0      else sa[at--] = m0[p--];
cbb      }
f2b      while (p >= 0) sa[at--] = m0[p--];
eb6      if (N%3==1) for (int i = 0; i < N; i++) sa[i] =
      sa[i+1];
cbb      }

938      suffix_array(const string& s_) : s(s_), n(s.size()),
      sa(n+3),
e62          cnt(n+1), rnk(n), lcp(n-1) {
9fe          vector<int> v(n+3);
f9b          for (int i = 0; i < n; i++) v[i] = i;
eba          radix(&v[0], &rnk[0], &s[0], n, 256);
e6d          int dif = 1;
830          for (int i = 0; i < n; i++)
419              v[rnk[i]] = dif += (i and s[rnk[i]] !=
      s[rnk[i-1]]);
7cf          if (n >= 2) rec(v, dif);
fb9          sa.resize(n);

76f          for (int i = 0; i < n; i++) rnk[sa[i]] = i;
892          for (int i = 0, k = 0; i < n; i++, k -= !!k) {
668              if (rnk[i] == n-1) {
5a4                  k = 0;
5e2                  continue;
cbb              }
39a              int j = sa[rnk[i]+1];
891              while (i+k < n and j+k < n and s[i+k] ==
      s[j+k]) k++;
825              lcp[rnk[i]] = k;
cbb          }
9ff          RMQ = rmq<int>(lcp);
cbb      }
      // hash ateh aqui (sem o RMQ): 1ff700

588      int query(int i, int j) {
d97          if (i == j) return n-i;
223          i = rnk[i], j = rnk[j];

```

```

c3a      return RMQ.query(min(i, j), max(i, j)-1);
cbb    }
71c    pair<int, int> next(int L, int R, int i, char c) {
024        int l = L, r = R+1;
40c        while (l < r) {
ee4            int m = (l+r)/2;
e7e            if (i+sa[m] >= n or s[i+sa[m]] < c) l = m+1;
ef3            else r = m;
cbb        }
575        if (l == R+1 or s[i+sa[l]] > c) return {-1, -1};
eb7        L = l;

9e2        l = L, r = R+1;
40c        while (l < r) {
ee4            int m = (l+r)/2;
1a1            if (i+sa[m] >= n or s[i+sa[m]] <= c) l = m+1;
ef3            else r = m;
cbb        }
56a        R = l-1;
e13        return {L, R};
cbb    }

// quantas vezes 't' ocorre em 's' - O(|t| log n)
66d    int count_substr(string& t) {
b2b        int L = 0, R = n-1;
c9d        for (int i = 0; i < t.size(); i++) {
de0            tie(L, R) = next(L, R, i, t[i]);
4fc            if (L == -1) return 0;
cbb        }
fbf        return R-L+1;
cbb    }

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

e68    ll dfs(int L, int R, int p) { // dfs na suffix tree
      chamado em pre ordem
c54        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'
f80 ll ans = (ext-p)*f(R-L+1);

// L eh terminal, e folha sse L == R
63c 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
548 (separadores < 'a', diferentes e inclusive
no final)
afc while (L <= R && (sa[L]+ext == n || s[sa[L]+ext]
< 'a')) {
f49 L++;
792 } */

add while (L <= R) {
5a8 int idx = L != R ? RMQ.index_query(L, R-1) :
-1;
5ef if (idx == -1 or lcp[idx] != ext) idx = R;

478 ans += dfs(L, idx, ext);
28d L = idx+1;
cbb }
ba7 return ans;
cbb }

// sum over substrings: computa, para toda substring
t distinta de s,
// \sum f(# ocorrencias de t em s) - 0 (n)
ca8 ll sos() { return dfs(0, n-1, 0); }
214 };

```

## 6.12 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

2fe struct dyn_sa {
3c9 struct node {
1d4 int sa, lcp;
ed1 node *l, *r, *p;
f0d int sz, mi;
17b node(int sa_, int lcp_, node* p_) : sa(sa_),
lcp(lcp_),
543 l(NULL), r(NULL), p(p_), sz(1), mi(lcp) {}
01e void update() {
58f sz = 1, mi = lcp;
bd7 if (l) sz += l->sz, mi = min(mi, l->mi);
a54 if (r) sz += r->sz, mi = min(mi, r->mi);
cbb }
214 };

bb7 node* root;
295 vector<ll> tag; // tag of a suffix (reversed id)
ac0 string s; // reversed

cf4 dyn_sa() : root(NULL) {}
e45 dyn_sa(string s_) : dyn_sa() {
ae4 reverse(s_.begin(), s_.end());
519 for (char c : s_) push_front(c);
cbb }

a86 ~dyn_sa() {
609 vector<node*> q = {root};
402 while (q.size()) {
e5d node* x = q.back(); q.pop_back();
ee9 if (!x) continue;
1c7 q.push_back(x->l), q.push_back(x->r);
bf0 delete x;
cbb }
cbb }

73c int size(node* x) { return x ? x->sz : 0; }
08e int mirror(int i) { return s.size()-1 - i; }
580 bool cmp(int i, int j) {
a29 if (s[i] != s[j]) return s[i] < s[j];
5b4 if (i == 0 or j == 0) return i < j;
988 return tag[i-1] < tag[j-1];

```



```

cbb      }
919      void fix_path(node* x) { while (x) x->update(), x =
x->p; }
245      void flatten(vector<node*>& v, node* x) {
8c8          if (!x) return;
e96          flatten(v, x->l);
2a2          v.push_back(x);
42d          flatten(v, x->r);
cbb      }
964      void build(vector<node*>& v, node*& x, node* p, int
L, int R, ll l, ll r) {
04c          if (L > R) return void(x = NULL);
331          int M = (L+R)/2;
3e3          ll m = (l+r)/2;
7e5          x = v[M];
63e          x->p = p;
bb3          tag[x->sa] = m;
ae0          build(v, x->l, x, L, M-1, l, m-1), build(v,
x->r, x, M+1, R, m+1, r);
ca8          x->update();
cbb      }
82f      void fix(node*& x, node* p, ll l, ll r) {
7f0          if (3*max(size(x->l), size(x->r)) <= 2*size(x))
return x->update();
3d1          vector<node*> v;
0cc          flatten(v, x);
ea9          build(v, x, p, 0, v.size()-1, l, r);
cbb      }
b19      node* next(node* x) {
728          if (x->r) {
a91              x = x->r;
347              while (x->l) x = x->l;
ea5              return x;
cbb          }
402          while (x->p and x->p->r == x) x = x->p;
137          return x->p;
cbb      }
b68      node* prev(node* x) {
e41          if (x->l) {
a26              x = x->l;
93c              while (x->r) x = x->r;
ea5              return x;

```

```

cbb      }
6a1          while (x->p and x->p->l == x) x = x->p;
137          return x->p;
cbb      }

4f7      int get_lcp(node* x, node* y) {
75a          if (!x or !y) return 0; // change default value
here
e51          if (s[x->sa] != s[y->sa]) return 0;
843          if (x->sa == 0 or y->sa == 0) return 1;
4d0          return 1 + query(mirror(x->sa-1),
mirror(y->sa-1));
cbb      }
ad6      void add_suf(node*& x, node* p, int id, ll l, ll r) {
91e          if (!x) {
8e3              x = new node(id, 0, p);
8e2              node *prv = prev(x), *nxt = next(x);
65d              int lcp_cur = get_lcp(prv, x), lcp_nxt =
get_lcp(x, nxt);
ca3              if (nxt) nxt->lcp = lcp_nxt, fix_path(nxt);
71f              x->lcp = lcp_cur;
7b4              tag[id] = (l+r)/2;
ca8              x->update();
505              return;
cbb          }
4a3          if (cmp(id, x->sa)) add_suf(x->l, x, id, l,
tag[x->sa]-1);
c3a          else add_suf(x->r, x, id, tag[x->sa]+1, r);
3db          fix(x, p, l, r);
cbb      }
ec2      void push_front(char c) {
cc7          s += c;
493          tag.push_back(-1);
05e          add_suf(root, NULL, s.size() - 1, 0, 1e18);
cbb      }

7f3      void rem_suf(node*& x, int id) {
6cf          if (x->sa != id) {
864              if (tag[id] < tag[x->sa]) return
rem_suf(x->l, id);
e6f              return rem_suf(x->r, id);
cbb          }

```

```

2cf      node* nxt = next(x);
09b      if (nxt) nxt->lcp = min(nxt->lcp, x->lcp),
fix_path(nxt);

b20      node *p = x->p, *tmp = x;
f3f      if (!x->l or !x->r) {
2fd          x = x->l ? x->l : x->r;
753          if (x) x->p = p;
9d9      } else {
7f7          for (tmp = x->l, p = x; tmp->r; tmp =
tmp->r) p = tmp;
f2a          x->sa = tmp->sa, x->lcp = tmp->lcp;
482          if (tmp->l) tmp->l->p = p;
14c          if (p->l == tmp) p->l = tmp->l;
a94          else p->r = tmp->l;
cbb      }
b5e      fix_path(p);
7c3      delete tmp;
cbb      }
15b      void pop_front() {
abe          if (!s.size()) return;
342          s.pop_back();
436          rem_suf(root, s.size());
c6e          tag.pop_back();
cbb      }

530      int query(node* x, ll l, ll r, ll a, ll b) {
e51          if (!x or tag[x->sa] == -1 or r < a or b < l)
return s.size();
ef5          if (a <= l and r <= b) return x->mi;
8eb          int ans = s.size();
e1f          if (a <= tag[x->sa] and tag[x->sa] <= b) ans =
min(ans, x->lcp);
d99          ans = min(ans, query(x->l, l, tag[x->sa]-1, a,
b));
261          ans = min(ans, query(x->r, tag[x->sa]+1, r, a,
b));
ba7          return ans;
cbb      }
588      int query(int i, int j) { // lcp(s[i..], s[j..])
209          if (i == j) return s.size() - i;
29e          ll a = tag[mirror(i)], b = tag[mirror(j)];

```

```

710          int ret = query(root, 0, 1e18, min(a, b)+1,
max(a, b));
edf          return ret;
cbb      }
// optional: get rank[i], sa[i] and lcp[i]
044      int rank(int i) {
396          i = mirror(i);
52f          node* x = root;
7c9          int ret = 0;
f4c          while (x) {
33e              if (tag[x->sa] < tag[i]) {
f9d                  ret += size(x->l)+1;
a91                  x = x->r;
eb5              } else x = x->l;
cbb          }
edf          return ret;
cbb      }
649      pair<int, int> operator[](int i) {
52f          node* x = root;
31e          while (1) {
d4d              if (i < size(x->l)) x = x->l;
4e6              else {
85f                  i -= size(x->l);
e03                  if (!i) return {mirror(x->sa), x->lcp};
040                  i--, x = x->r;
cbb              }
cbb          }
cbb      }
214 };

```

## 6.13 Trie

```

// trie T() constroi uma trie para o alfabeto das letras
minuscucas
// 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

ab5 struct trie {
e1a     vector<vector<int>> to;
450     vector<int> end, pref;
af0     int sigma; char norm;
bb1     trie(int sigma_=26, char norm_='a') : sigma(sigma_),
        norm(norm_) {
58a         to = {vector<int>(sigma)};
86e         end = {0}, pref = {0};
cbb     }
64e     void insert(string s) {
c67         int x = 0;
7e7         for(auto c : s) {
008             int &nxt = to[x][c-norm];
dd7             if(!nxt) {
0aa                 nxt = to.size();
526                 to.push_back(vector<int>(sigma));
770                 end.push_back(0), pref.push_back(0);
cbb             }
827             x = nxt, pref[x]++;
cbb         }
e4e         end[x]++;
cbb     }
6b2     void erase(string s) {
c67         int x = 0;
b4f         for(char c : s) {
008             int &nxt = to[x][c-norm];
10c             x = nxt, pref[x]--;
d8e             if(!pref[x]) nxt = 0;
cbb         }
bf0         end[x]--;
cbb     }
aee     int find(string s) {
c67         int x = 0;
7e7         for(auto c : s) {
2ec             x = to[x][c-norm];
a66             if(!x) return 0;
cbb         }
ea5         return x;

```

```

cbb     }
839     int count_pref(string s) {
e2f         return pref[find(s)];
cbb     }
214 };

```

## 7 Primitivas

### 7.1 Aritmetica Modular

```

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

429 template<int p> struct mod_int {
02c     ll pow(ll b, ll e) {
a63         if (e == 0) return 1;
630         ll r = pow(b*b%p, e/2);
475         if (e%2 == 1) r = (r*b)%p;
4c1         return r;
cbb     }
ae3     ll inv(ll b) { return pow(b, p-2); }

4d7     using m = mod_int;
d93     int v;
fe0     mod_int() : v(0) {}
e12     mod_int(ll v_) {
019         if (v_ >= p or v_ <= -p) v_ %= p;
bc6         if (v_ < 0) v_ += p;
2e7         v = v_;
cbb     }
74d     m& operator+=(const m &a) {
2fd         v += a.v;
ba5         if (v >= p) v -= p;
357         return *this;
cbb     }
eff     m& operator-=(const m &a) {
8b4         v -= a.v;
cc8         if (v < 0) v += p;
357         return *this;
cbb     }

```

```

4c4      m& operator*=(const m &a) {
8a5          v = v * ll(a.v) % p;
357          return *this;
cbb      }
3f9      m& operator/=(const m &a) {
5d6          v = v* inv(a.v) % p;
357          return *this;
cbb      }
d65      m operator-(){ return m(-v); }
b3e      m& operator^=(ll e) {
06d          if (e < 0){
6e2              v = inv(v);
00c              e = -e;
cbb          }
ebf          v = pow(v, e%(p-1));
357          return *this;
cbb      }
423      bool operator==(const m &a) { return v == a.v; }
69f      bool operator!=(const m &a) { return v != a.v; }

1c6      friend istream &operator>>(istream &in, m& a) {
d1c          ll val; in >> val;
d48          a = m(val);
091          return in;
cbb      }
44f      friend ostream &operator<<(ostream &out, m a) {
5a0          return out << a.v;
cbb      }
399      friend m operator+(m a, m b) { return a+=b; }
f9e      friend m operator-(m a, m b) { return a-=b; }
9c1      friend m operator*(m a, m b) { return a*=b; }
51b      friend m operator/(m a, m b) { return a/=b; }
08f      friend m operator^(m a, ll e) { return a^=e; }
214 };

055 typedef mod_int<(int)1e9+7> mint;

```

## 7.2 Big Integer

```

// Complexidades: (para n digitos)
// Soma, subtracao, comparacao - O(n)
// Multiplicacao - O(n log(n))

```

```

// Divisao, resto - O(n^2)

864 struct bint {
669     static const int BASE = 1e9;
990     vector<int> v;
3bd     bool neg;

609     bint() : neg(0) {}
d53     bint(int val) : bint() { *this = val; }
e8f     bint(long long val) : bint() { *this = val; }

a0f     void trim() {
f42         while (v.size() and v.back() == 0) v.pop_back();
df8         if (!v.size()) neg = 0;
cbb     }

        // converter de/para string | cin/cout
294     bint(const char* s) : bint() {
        from_string(string(s)); }
548     bint(const string& s) : bint() { from_string(s); }
4ab     void from_string(const string& s) {
0a6         v.clear(), neg = 0;
d72         int ini = 0;
8e2         while (ini < s.size() and (s[ini] == '-' or
s[ini] == '+' or s[ini] == '0'))
71d             if (s[ini++] == '-') neg = 1;
883         for (int i = s.size()-1; i >= ini; i -= 9) {
05e             int at = 0;
5b1             for (int j = max(ini, i - 8); j <= i; j++)
                at = 10*at + (s[j]-'0');
1fd             v.push_back(at);
cbb         }
df8         if (!v.size()) neg = 0;
cbb     }

2ff     string to_string() const {
8be         if (!v.size()) return "0";
793         string ret;
73e         if (neg) ret += '-';
3e9         for (int i = v.size()-1; i >= 0; i--) {
582             string at = ::to_string(v[i]);
ced             int add = 9 - at.size();
75e             if (i+1 < v.size()) for (int j = 0; j < add;

```

```

j++) ret += '0';
f9f         ret += at;
cbb     }
edf         return ret;
cbb     }
d2f     friend istream& operator>>(istream& in, bint& val) {
eb6         string s; in >> s;
966         val = s;
091         return in;
cbb     }
99d     friend ostream& operator<<(ostream& out, const bint&
val) {
8b9         string s = val.to_string();
396         out << s;
fe8         return out;
cbb     }

// operators
60a     friend bint abs(bint val) {
c5f         val.neg = 0;
d94         return val;
cbb     }
bee     friend bint operator-(bint val) {
815         if (val != 0) val.neg ^= 1;
d94         return val;
cbb     }
41f     bint& operator=(const bint& val) { v = val.v, neg =
val.neg; return *this; }
249     bint& operator=(long long val) {
0a6         v.clear(), neg = 0;
3a6         if (val < 0) neg = 1, val *= -1;
fdc         for (; val; val /= BASE) v.push_back(val % BASE);
357         return *this;
cbb     }
3bd     int cmp(const bint& r) const { // menor: -1 | igual:
0 | maior: 1
b14         if (neg != r.neg) return neg ? -1 : 1;
0bb         if (v.size() != r.v.size()) {
ff7             int ret = v.size() < r.v.size() ? -1 : 1;
91b             return neg ? -ret : ret;
cbb         }
478         for (int i = int(v.size())-1; i >= 0; i--) {

```

```

405             if (v[i] != r.v[i]) {
2e5                 int ret = v[i] < r.v[i] ? -1 : 1;
91b                 return neg ? -ret : ret;
cbb             }
cbb         }
bb3         return 0;
cbb     }
152     friend bool operator<(const bint& l, const bint& r)
{ return l.cmp(r) == -1; }
c7a     friend bool operator>(const bint& l, const bint& r)
{ return l.cmp(r) == 1; }
edd     friend bool operator<=(const bint& l, const bint& r)
{ return l.cmp(r) <= 0; }
954     friend bool operator>=(const bint& l, const bint& r)
{ return l.cmp(r) >= 0; }
a67     friend bool operator==(const bint& l, const bint& r)
{ return l.cmp(r) == 0; }
10b     friend bool operator!=(const bint& l, const bint& r)
{ return l.cmp(r) != 0; }

38e     bint& operator +=(const bint& r) {
6bf         if (!r.v.size()) return *this;
a93         if (neg != r.neg) return *this -= -r;
256         for (int i = 0, c = 0; i < r.v.size() or c; i++)
{
e28             if (i == v.size()) v.push_back(0);
08f             v[i] += c + (i < r.v.size() ? r.v[i] : 0);
baa             if ((c = v[i] >= BASE)) v[i] -= BASE;
cbb         }
357         return *this;
cbb     }
54c     friend bint operator+(bint a, const bint& b) {
return a += b; }
9c8     bint& operator -=(const bint& r) {
6bf         if (!r.v.size()) return *this;
524         if (neg != r.neg) return *this += -r;
358         if ((!neg and *this < r) or (neg and r < *this))
{
b10             *this = r - *this;
a10             neg ^= 1;
357             return *this;
cbb         }

```

```

256         for (int i = 0, c = 0; i < r.v.size() or c; i++)
{
9ef             v[i] -= c + (i < r.v.size() ? r.v[i] : 0);
c8c             if ((c = v[i] < 0)) v[i] += BASE;
cbb         }
0eb         trim();
357         return *this;
cbb     }
f44     friend bint operator-(bint a, const bint& b) {
        return a -= b; }

    // operators de * / %
6b0     bint& operator *=(int val) {
bca         if (val < 0) val *= -1, neg ^= 1;
566         for (int i = 0, c = 0; i < v.size() or c; i++) {
e28             if (i == v.size()) v.push_back(0);
352             long long at = (long long) v[i] * val + c;
6a3             v[i] = at % BASE;
b3d             c = at / BASE;
cbb         }
0eb         trim();
357         return *this;
cbb     }
480     friend bint operator *(bint a, int b) { return a *=
        b; }
d5c     friend bint operator *(int a, bint b) { return b *=
        a; }
13b     using cplx = complex<double>;
bfb     void fft(vector<cplx>& a, bool f, int N,
        vector<int>& rev) const {
bc7         for (int i = 0; i < N; i++) if (i < rev[i])
            swap(a[i], a[rev[i]]);
bad         vector<cplx> roots(N);
192         for (int n = 2; n <= N; n *= 2) {
4e9             const static double PI = acos(-1);
71a             for (int i = 0; i < n/2; i++) {
40d                 double alpha = (2*PI*i)/n;
1a1                 if (f) alpha = -alpha;
3f6                 roots[i] = cplx(cos(alpha), sin(alpha));
cbb             }
3e9             for (int pos = 0; pos < N; pos += n)
898                 for (int l = pos, r = pos+n/2, m = 0; m

```

```

        < n/2; l++, r++, m++) {
297             auto t = roots[m]*a[r];
254             a[r] = a[l] - t;
b8f             a[l] = a[l] + t;
cbb         }
cbb     }
3f1     if (!f) return;
08b     auto invN = cplx(1)/cplx(N);
873     for (int i = 0; i < N; i++) a[i] *= invN;
cbb }
0e0     vector<long long> convolution(const vector<int>& a,
        const vector<int>& b) const {
ff9         vector<cplx> l(a.begin(), a.end()), r(b.begin(),
        b.end());
996         int ln = l.size(), rn = r.size(), N = ln+rn+1, n
        = 1, log_n = 0;
821         while (n <= N) n <= 1, log_n++;
808         vector<int> rev(n);
603         for (int i = 0; i < n; i++) {
434             rev[i] = 0;
f44             for (int j = 0; j < log_n; j++) if (i>>j&1)
4ff                 rev[i] |= 1 << (log_n-1-j);
cbb         }
230         l.resize(n), r.resize(n);
a89         fft(l, false, n, rev), fft(r, false, n, rev);
917         for (int i = 0; i < n; i++) l[i] *= r[i];
88b         fft(l, true, n, rev);
7ae         vector<long long> ret;
c14         for (auto& i : l) ret.push_back(round(i.real()));
edf         return ret;
cbb     }
633     vector<int> convert_base(const vector<int>& a, int
        from, int to) const {
498         static vector<long long> pot(10, 1);
671         if (pot[1] == 1) for (int i = 1; i < 10; i++)
            pot[i] = 10*pot[i-1];
4b8         vector<int> ret;
156         long long at = 0;
608         int digits = 0;
941         for (int i : a) {
412             at += i * pot[digits];
035             digits += from;

```

```

684         while (digits >= to) {
0c8             ret.push_back(at % pot[to]);
cf9             at /= pot[to];
fd4             digits -= to;
cbb         }
cbb     }
944     ret.push_back(at);
384     while (ret.size() and ret.back() == 0)
        ret.pop_back();
edf     return ret;
cbb     }
edb     bint operator*(const bint& r) const { // O(n log(n))
2af         bint ret;
968         ret.neg = neg ^ r.neg;
d5d         auto conv = convolution(convert_base(v, 9, 4),
        convert_base(r.v, 9, 4));
a0e         long long c = 0;
a74         for (auto i : conv) {
f6d             long long at = i+c;
4cb             ret.v.push_back(at % 10000);
a25             c = at / 10000;
cbb         }
3cb         for (; c; c /= 10000) ret.v.push_back(c%10000);
0e2         ret.v = convert_base(ret.v, 4, 9);
25c         if (!ret.v.size()) ret.neg = 0;
edf         return ret;
cbb     }
359     bint& operator*=(const bint& r) { return *this =
*this * r; };
9a3     bint& operator/=(int val) {
d9a         if (val < 0) neg ^= 1, val *= -1;
f18         for (int i = int(v.size())-1, c = 0; i >= 0;
i--) {
2a7             long long at = v[i] + c * (long long) BASE;
e02             v[i] = at / val;
fb1             c = at % val;
cbb         }
0eb         trim();
357         return *this;
cbb     }
e74     friend bint operator/(bint a, int b) { return a /=
b; }

```

```

4a9     int operator %=(int val) {
23b         if (val < 0) val *= -1;
156         long long at = 0;
f31         for (int i = int(v.size())-1; i >= 0; i--)
1b3             at = (BASE * at + v[i]) % val;
d22         if (neg) at *= -1;
ce6         return at;
cbb     }
2fb     friend int operator%(bint a, int b) { return a %= b;
}
13b     friend pair<bint, bint> divmod(const bint& a_, const
bint& b_) { // O(n^2)
611         if (a_ == 0) return {0, 0};
d8a         int norm = BASE / (b_.v.back() + 1);
b4e         bint a = abs(a_) * norm;
027         bint b = abs(b_) * norm;
14d         bint q, r;
c91         for (int i = a.v.size() - 1; i >= 0; i--) {
b71             r *= BASE, r += a.v[i];
4ff             long long upper = b.v.size() < r.v.size() ?
r.v[b.v.size()] : 0;
86d             int lower = b.v.size() - 1 < r.v.size() ?
r.v[b.v.size() - 1] : 0;
431             int d = (upper * BASE + lower) / b.v.back();
5d4             r -= b*d;
30f             while (r < 0) r += b, d--; // roda O(1) vezes
738             q.v.push_back(d);
cbb         }
a48         reverse(q.v.begin(), q.v.end());
ae2         q.neg = a_.neg ^ b_.neg;
88b         r.neg = a_.neg;
8e5         q.trim(), r.trim();
0ef         return {q, r / norm};
cbb     }
1d8     bint operator/(const bint& val) { return
divmod(*this, val).first; }
7f9     bint& operator/=(const bint& val) { return *this =
*this / val; }
1f9     bint operator%(const bint& val) { return
divmod(*this, val).second; }
df5     bint& operator%=(const bint& val) { return *this =
*this % val; }

```

```
214 };
```

## 7.3 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

fda struct graphic_matroid {
5da     int n, m, t;
32c     vector<array<int, 2>> edges;
789     vector<vector<int>> g;
62e     vector<int> comp, in, out;
513     graphic_matroid(int n_, vector<array<int, 2>> edges_)
a1f         : n(n_), m(edges_.size()), edges(edges_), g(n),
        comp(n), in(n), out(n) {}
315     void dfs(int u) {
ab8         in[u] = t++;
17d         for (auto v : g[u]) if (in[v] == -1)
863             comp[v] = comp[u], dfs(v);
677         out[u] = t;
cbb     }
945     void build(vector<int> I) {
a34         t = 0;
741         for (int u = 0; u < n; u++) g[u].clear(), in[u]
```

```
        = -1;
667         for (int e : I) {
d00             auto [u, v] = edges[e];
125             g[u].push_back(v), g[v].push_back(u);
cbb         }
809         for (int u = 0; u < n; u++) if (in[u] == -1)
a7d             comp[u] = u, dfs(u);
cbb     }
f31     bool is_ancestor(int u, int v) {
a68         return in[u] <= in[v] and in[v] < out[u];
cbb     }
e6b     bool oracle(int e) {
453         return comp[edges[e][0]] != comp[edges[e][1]];
cbb     }
f75     bool oracle(int e, int f) {
574         if (oracle(f)) return true;
622         int u = edges[e][in[edges[e][0]] <
in[edges[e][1]]];
ff2         return is_ancestor(u, edges[f][0]) !=
is_ancestor(u, edges[f][1]);
cbb     }
214 };

// 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

994 struct partition_matroid {
501     vector<int> cap, color, d;
608     partition_matroid(vector<int> cap_, vector<int>
color_)
04d         : cap(cap_), color(color_), d(cap.size()) {}
945     void build(vector<int> I) {
def         fill(d.begin(), d.end(), 0);
```



```

e9d         for (int u : I) d[color[u]]++;
cbb     }
514     bool oracle(int u) {
0a1         return d[color[u]] < cap[color[u]];
cbb     }
f7f     bool oracle(int u, int v) {
2f7         return color[u] == color[v] or oracle(v);
cbb     }
214 };

// 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
132 template<typename Matroid1, typename Matroid2>
801 vector<int> matroid_intersection(int n, Matroid1 M1,
    Matroid2 M2) {
f5b     vector<bool> b(n);
a64     vector<int> I[2];
a8b     bool converged = false;
0c1     while (!converged) {
742         I[0].clear(), I[1].clear();
99d         for (int u = 0; u < n; u++) I[b[u]].push_back(u);

09d         M1.build(I[1]), M2.build(I[1]);
289         vector<bool> target(n), pushed(n);
26a         queue<int> q;
5c5         for (int u : I[0]) {
2b2             target[u] = M2.oracle(u);
c1b             if (M1.oracle(u)) pushed[u] = true,
                q.push(u);
cbb         }
3fe         vector<int> p(n, -1);
07a         converged = true;
402         while (q.size()) {
be1             int u = q.front(); q.pop();
5c6             if (target[u]) {

```

```

101             converged = false;
c32             for (int v = u; v != -1; v = p[v]) b[v]
                = !b[v];
c2b             break;
cbb         }
e78         for (int v : I[!b[u]]) if (!pushed[v]) {
34d             if ((b[u] and M1.oracle(u, v)) or (b[v]
                and M2.oracle(v, u)))
bae                 p[v] = u, pushed[v] = true,
                q.push(v);
cbb         }
cbb     }
cbb     }
b68     return I[1];
cbb }

// 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

42a template<typename T, typename Matroid1, typename
    Matroid2>
2b5 vector<int> weighted_matroid_intersection(int n,
    vector<T> w, Matroid1 M1, Matroid2 M2) {
6c9     vector<bool> b(n), target(n), is_inside(n);
563     vector<int> I[2], from(n);
e35     vector<pair<T, int>> d(n);
169     auto check_edge = [&](int u, int v) {
249         return (b[u] and M1.oracle(u, v)) or (b[v] and
            M2.oracle(v, u));
214     };
667     while (true) {

```

```

742     I[0].clear(), I[1].clear();
99d     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
09d     M1.build(I[1]), M2.build(I[1]);
687     for (int u = 0; u < n; u++) {
ea5         target[u] = false, is_inside[u] = false,
        from[u] = -1;
961         d[u] = {numeric_limits<T>::max(), INF};
cbb     }
8d3     deque<T> q;
476     sort(I[0].begin(), I[0].end(), [&](int i, int
j){ return w[i] < w[j]; });
5c5     for (int u : I[0]) {
2b2         target[u] = M2.oracle(u);
5a7         if (M1.oracle(u)) {
4ef             if (is_inside[u]) continue;
7cc             d[u] = {w[u], 0};
427             if (!q.empty() and d[u] > d[q.front()])
                q.push_back(u);
655             else q.push_front(u);
4ae             is_inside[u] = true;
cbb         }
cbb     }
402     while (q.size()) {
97a         int u = q.front(); q.pop_front();
6f3         is_inside[u] = false;
57a         for (int v : I[!b[u]]) if (check_edge(u, v))
        {
9de             pair<T, int> nd(d[u].first + w[v],
                d[u].second + 1);
61b             if (nd < d[v]) {
6ac                 from[v] = u, d[v] = nd;
bd7                 if (is_inside[v]) continue;
eec                 if (q.size() and d[v] >
                    d[q.front()]) q.push_back(v);
275                 else q.push_front(v);
587                 is_inside[v] = true;
cbb             }
cbb         }
cbb     }
cbb     pair<T, int> mini =

```

```

        pair(numeric_limits<T>::max(), INF);
489         int targ = -1;
259         for (int u : I[0]) if (target[u] and d[u] < mini)
2b9             mini = d[u], targ = u;
e14         if (targ != -1) for (int u = targ; u != -1; u =
            from[u])
d89             b[u] = !b[u], w[u] *= -1;
f97         else break;
cbb     }
b68     return I[1];
cbb }

```

## 7.4 Primitivas de fracao

// Funciona com o Big Int  
// cdb445

```

a4e template<typename T = int> struct frac {
a40     T num, den;
e3f     template<class U, class V>
61d     frac(U num_ = 0, V den_ = 1) : num(num_), den(den_) {
bad         assert(den != 0);
583         if (den < 0) num *= -1, den *= -1;
a51         T g = gcd(abs(num), den);
572         num /= g, den /= g;
cbb     }

51f     friend bool operator<(const frac& l, const frac& r) {
fa0         return l.num * r.den < r.num * l.den;
cbb     }
4b5     friend frac operator+(const frac& l, const frac& r) {
b61         return {l.num*r.den + l.den*r.num, l.den*r.den};
cbb     }
74d     friend frac operator-(const frac& l, const frac& r) {
2cd         return {l.num*r.den - l.den*r.num, l.den*r.den};
cbb     }
c80     friend frac operator*(const frac& l, const frac& r) {
510         return {l.num*r.num, l.den*r.den};
cbb     }
a1b     friend frac operator/(const frac& l, const frac& r) {
8f3         return {l.num*r.den, l.den*r.num};
cbb     }

```

```

012     friend ostream& operator<<(ostream& out, frac f) {
37a         out << f.num << '/' << f.den;
fe8         return out;
cbb     }
214 };

```

## 7.5 Primitivas de matriz - exponenciacao

```

// d05c24

945 #define MODULAR false
5ed template<typename T> struct matrix : vector<vector<T>> {
14e     int n, m;

30f     void print() {
603         for (int i = 0; i < n; i++) {
70f             for (int j = 0; j < m; j++) cout <<
(*this)[i][j] << " ";
1fb             cout << endl;
cbb         }
cbb     }

aa3     matrix(int n_, int m_, bool ident = false) :
b14         vector<vector<T>>(n_, vector<T>(m_, 0)),
n(n_), m(m_) {
94e         if (ident) {
df7             assert(n == m);
a89             for (int i = 0; i < n; i++) (*this)[i][i] =
1;
cbb         }
cbb     }

b83     matrix(const vector<vector<T>>& c) :
vector<vector<T>>(c),
a3d         n(c.size()), m(c[0].size()) {}
efc     matrix(const initializer_list<initializer_list<T>>&
c) {
f7e         vector<vector<T>> val;
212         for (auto& i : c) val.push_back(i);
303         *this = matrix(val);
cbb     }

388     matrix<T> operator*(matrix<T>& r) {

```

```

1e2         assert(m == r.n);
82c         matrix<T> M(n, r.m);
d69         for (int i = 0; i < n; i++) for (int k = 0; k <
m; k++)
df4             for (int j = 0; j < r.m; j++) {
e34                 T add = (*this)[i][k] * r[k][j];
f98 #if MODULAR
d41 #warning Usar matrix<ll> e soh colocar valores em [0,
MOD) na matriz!
8b6                 M[i][j] += add%MOD;
983                 if (M[i][j] >= MOD) M[i][j] -= MOD;
8c1 #else
7bb                 M[i][j] += add;
f2e #endif
cbb             }
474         return M;
cbb     }

528     matrix<T> operator^(ll e){
f10         matrix<T> M(n, n, true), at = *this;
c87         while (e) {
2e2             if (e&1) M = M*at;
cc2             e >>= 1;
c80             at = at*at;
cbb         }
474         return M;
cbb     }

582     void apply_transform(matrix M, ll e){
1c3         auto& v = *this;
c87         while (e) {
9ba             if (e&1) v = M*v;
cc2             e >>= 1;
419             M = M*M;
cbb         }
cbb     }
214 };

```

## 7.6 Primitivas Geometricas

```

c83 typedef double ld;
e3b const ld DINF = 1e18;
43a const ld pi = acos(-1.0);
107 const ld eps = 1e-9;

```

```

b32 #define sq(x) ((x)*(x))

d97 bool eq(ld a, ld b) {
ba0     return abs(a - b) <= eps;
cbb }

    // a8b7d6
b2a struct pt { // ponto
c1e     ld x, y;
3dd     pt(ld x_ = 0, ld y_ = 0) : x(x_), y(y_) {}
5bc     bool operator < (const pt p) const {
059         if (!eq(x, p.x)) return x < p.x;
f98         if (!eq(y, p.y)) return y < p.y;
bb3         return 0;
cbb     }
a83     bool operator == (const pt p) const {
ed0         return eq(x, p.x) and eq(y, p.y);
cbb     }
cb9     pt operator + (const pt p) const { return pt(x+p.x,
y+p.y); }
a24     pt operator - (const pt p) const { return pt(x-p.x,
y-p.y); }
4a8     pt operator * (const ld c) const { return pt(x*c ,
y*c ); }
a60     pt operator / (const ld c) const { return pt(x/c ,
y/c ); }
3b6     ld operator * (const pt p) const { return x*p.x +
y*p.y; }
6df     ld operator ^ (const pt p) const { return x*p.y -
y*p.x; }
5ed     friend istream& operator >> (istream& in, pt& p) {
e37         return in >> p.x >> p.y;
cbb     }
214 };

    // 7ab617
b3a struct line { // reta
730     pt p, q;
0d6     line() {}
4b8     line(pt p_, pt q_) : p(p_), q(q_) {}
8d7     friend istream& operator >> (istream& in, line& r) {

```

```

4cb         return in >> r.p >> r.q;
cbb     }
214 };

    // PONTO & VETOR

    // c684fb
364 ld dist(pt p, pt q) { // distancia
5f3     return hypot(p.y - q.y, p.x - q.x);
cbb }

    // 80f2b6
9d7 ld dist2(pt p, pt q) { // quadrado da distancia
f24     return sq(p.x - q.x) + sq(p.y - q.y);
cbb }

    // cf7f33
483 ld norm(pt v) { // norma do vetor
490     return dist(pt(0, 0), v);
cbb }

    // 404df7
589 ld angle(pt v) { // angulo do vetor com o eixo x
587     ld ang = atan2(v.y, v.x);
6f8     if (ang < 0) ang += 2*pi;
19c     return ang;
cbb }

    // 1b1d4a
298 ld sarea(pt p, pt q, pt r) { // area com sinal
606     return ((q-p)^(r-q))/2;
cbb }

    // 98c42f
e32 bool col(pt p, pt q, pt r) { // se p, q e r sao colin.
e7d     return eq(sarea(p, q, r), 0);
cbb }

    // 85d09d
0cd bool ccw(pt p, pt q, pt r) { // se p, q, r sao ccw
fa7     return sarea(p, q, r) > eps;
cbb }

```

```

// 41a7b4
1ef pt rotate(pt p, ld th) { // rotaciona o ponto th radianos
e5c     return pt(p.x * cos(th) - p.y * sin(th),
ff1         p.x * sin(th) + p.y * cos(th));
cbb }

// e4ad5e
ab1 pt rotate90(pt p) { // rotaciona 90 graus
a0d     return pt(-p.y, p.x);
cbb }

// RETA

// 0fb984
edc bool isvert(line r) { // se r eh vertical
87d     return eq(r.p.x, r.q.x);
cbb }

// 726d68
099 bool isinseg(pt p, line r) { // se p pertence ao seg de r
f65     pt a = r.p - p, b = r.q - p;
b04     return eq((a ^ b), 0) and (a * b) < eps;
cbb }

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

// 2329fe
256 pt proj(pt p, line r) { // projecao do ponto p na reta r
bea     if (r.p == r.q) return r.p;
97a     r.q = r.q - r.p; p = p - r.p;
9f8     pt proj = r.q * ((p*r.q) / (r.q*r.q));
2cd     return proj + r.p;
cbb }

// 111fd2
d5c pt inter(line r, line s) { // r inter s
146     if (eq((r.p - r.q) ^ (s.p - s.q), 0)) return

```

```

    pt(DINF, DINF);
205     r.q = r.q - r.p, s.p = s.p - r.p, s.q = s.q - r.p;
543     return r.q * get_t(r.q, s) + r.p;
cbb }

// 35998c
676 bool interseg(line r, line s) { // se o seg de r
    intersecta o seg de s
19b     if (isinseg(r.p, s) or isinseg(r.q, s)
c21         or isinseg(s.p, r) or isinseg(s.q, r)) return 1;

9fa     return ccw(r.p, r.q, s.p) != ccw(r.p, r.q, s.q) and
413         ccw(s.p, s.q, r.p) != ccw(s.p, s.q, r.q);
cbb }

// 1b72e1
fcb ld disttoline(pt p, line r) { // distancia do ponto a
    reta
89a     return 2 * abs(sarea(p, r.p, r.q)) / dist(r.p, r.q);
cbb }

// 3679c0
bcc ld disttoseg(pt p, line r) { // distancia do ponto ao seg
73d     if ((r.q - r.p)*(p - r.p) < 0) return dist(r.p, p);
951     if ((r.p - r.q)*(p - r.q) < 0) return dist(r.q, p);
a19     return disttoline(p, r);
cbb }

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

349     ld ret = DINF;
341     ret = min(ret, disttoseg(a.p, b));
ceb     ret = min(ret, disttoseg(a.q, b));
093     ret = min(ret, disttoseg(b.p, a));
448     ret = min(ret, disttoseg(b.q, a));

edf     return ret;
cbb }

// POLIGONO

```

```

// corta poligono com a reta r deixando os pontos p tal
// que
// ccw(r.p, r.q, p)
// 2538f9
1a9 vector<pt> cut_polygon(vector<pt> v, line r) { // 0(n)
8af     vector<pt> ret;
8a4     for (int j = 0; j < v.size(); j++) {
dac         if (ccw(r.p, r.q, v[j])) ret.push_back(v[j]);
dce         if (v.size() == 1) continue;
030         line s(v[j], v[(j+1)%v.size()]);
ae3         pt p = inter(r, s);
a3d         if (isinseg(p, s)) ret.push_back(p);
cbb     }
8a1     ret.erase(unique(ret.begin(), ret.end()), ret.end());
24d     if (ret.size() > 1 and ret.back() == ret[0])
        ret.pop_back();
edf     return ret;
cbb }

// distancia entre os retangulos a e b (lados paralelos
// aos eixos)
// assume que ta representado (inferior esquerdo,
// superior direito)
// 630253
5f5 ld dist_rect(pair<pt, pt> a, pair<pt, pt> b) {
080     ld hor = 0, vert = 0;
34b     if (a.second.x < b.first.x) hor = b.first.x -
        a.second.x;
f5f     else if (b.second.x < a.first.x) hor = a.first.x -
        b.second.x;
4fd     if (a.second.y < b.first.y) vert = b.first.y -
        a.second.y;
80a     else if (b.second.y < a.first.y) vert = a.first.y -
        b.second.y;
96f     return dist(pt(0, 0), pt(hor, vert));
cbb }

// 5df9cf
13d ld polarea(vector<pt> v) { // area do poligono
9c5     ld ret = 0;
c6e     for (int i = 0; i < v.size(); i++)

```

```

80f         ret += sarea(pt(0, 0), v[i], v[(i + 1) %
v.size()]);
d03     return abs(ret);
cbb }

// se o ponto ta dentro do poligono: retorna 0 se ta
// fora,
// 1 se ta no interior e 2 se ta na borda
// a6423f
8e7 int inpol(vector<pt>& v, pt p) { // 0(n)
8de     int qt = 0;
f14     for (int i = 0; i < v.size(); i++) {
bda         if (p == v[i]) return 2;
6af         int j = (i+1)%v.size();
e38         if (eq(p.y, v[i].y) and eq(p.y, v[j].y)) {
97f             if ((v[i]-p)*(v[j]-p) < eps) return 2;
5e2             continue;
cbb         }
388         bool baixo = v[i].y+eps < p.y;
464         if (baixo == (v[j].y+eps < p.y)) continue;
366         auto t = (p-v[i])^(v[j]-v[i]);
1b4         if (eq(t, 0)) return 2;
839         if (baixo == (t > eps)) qt += baixo ? 1 : -1;
cbb     }
b84     return qt != 0;
cbb }

// c58350
6ff bool interpol(vector<pt> v1, vector<pt> v2) { // se dois
        poligonos se intersectam - 0(n*m)
7d1     int n = v1.size(), m = v2.size();
c36     for (int i = 0; i < n; i++) if (inpol(v2, v1[i]))
        return 1;
ab8     for (int i = 0; i < n; i++) if (inpol(v1, v2[i]))
        return 1;
523     for (int i = 0; i < n; i++) for (int j = 0; j < m;
j++)
0c8         if (interseg(line(v1[i], v1[(i+1)%n]),
        line(v2[j], v2[(j+1)%m]))) return 1;
bb3     return 0;
cbb }

```

```

// 12559f
494 ld distpol(vector<pt> v1, vector<pt> v2) { // distancia
entre poligonos
f6b     if (interpol(v1, v2)) return 0;

349     ld ret = DINF;

1c8     for (int i = 0; i < v1.size(); i++) for (int j = 0;
j < v2.size(); j++)
6c2         ret = min(ret, distseg(line(v1[i], v1[(i + 1) %
v1.size()]),
9d9             line(v2[j], v2[(j + 1) %
v2.size()])));
edf     return ret;
cbb }

// 32623c
138 vector<pt> convex_hull(vector<pt> v) { // convex hull -
O(n log(n))
52d     if (v.size() <= 1) return v;
526     vector<pt> l, u;
fca     sort(v.begin(), v.end());
f14     for (int i = 0; i < v.size(); i++) {
543         while (l.size() > 1 and !ccw(l[l.size()-2],
l.back(), v[i]))
364             l.pop_back();
c35         l.push_back(v[i]);
cbb     }
3e9     for (int i = v.size() - 1; i >= 0; i--) {
2eb         while (u.size() > 1 and !ccw(u[u.size()-2],
u.back(), v[i]))
7a8             u.pop_back();
a95         u.push_back(v[i]);
cbb     }
cfc     l.pop_back(); u.pop_back();
82b     for (pt i : u) l.push_back(i);
792     return l;
cbb }

483 struct convex_pol {
f50     vector<pt> pol;

```

```

// nao pode ter ponto colinear no convex hull
convex_pol() {}
a04 convex_pol(vector<pt> v) : pol(convex_hull(v)) {}

// se o ponto ta dentro do hull - O(log(n))
// 800813
8af bool is_inside(pt p) {
eae     if (pol.size() == 1) return p == pol[0];
67f     int l = 1, r = pol.size();
40c     while (l < r) {
ee4         int m = (l+r)/2;
48f         if (ccw(p, pol[0], pol[m])) l = m+1;
ef3         else r = m;
cbb     }
00a     if (l == 1) return isinseg(p, line(pol[0],
pol[1]));
9e7     if (l == pol.size()) return false;
1c0     return !ccw(p, pol[l], pol[l-1]);
cbb }

// ponto extremo em relacao a cmp(p, q) = p mais
extremo q
// (copiado de
https://github.com/gustavoM32/caderno-zika)
// 56ccd2
719 int extreme(const function<bool(pt, pt)>& cmp) {
b1c     int n = pol.size();
4a2     auto extr = [&](int i, bool& cur_dir) {
22a         cur_dir = cmp(pol[(i+1)%n], pol[i]);
61a         return !cur_dir and !cmp(pol[(i+n-1)%n],
pol[i]);
214     };
63d     bool last_dir, cur_dir;
a0d     if (extr(0, last_dir)) return 0;
993     int l = 0, r = n;
ead     while (l+1 < r) {
ee4         int m = (l+r)/2;
f29         if (extr(m, cur_dir)) return m;
44a         bool rel_dir = cmp(pol[m], pol[l]);
b18         if ((!last_dir and cur_dir) or
261             (last_dir == cur_dir and rel_dir ==
cur_dir)) {
8a6             l = m;

```

```

1f1         last_dir = cur_dir;
b6c     } else r = m;
cbb     }
792     return l;
cbb     }
316     int max_dot(pt v) {
ec1         return extreme([&](pt p, pt q) { return p*v >
q*v; });
cbb     }
a54     pair<int, int> tangents(pt p) {
08c         auto L = [&](pt q, pt r) { return ccw(p, q, r);
};
422         auto R = [&](pt q, pt r) { return ccw(p, r, q);
};
fa8         return {extreme(L), extreme(R)};
cbb     }
214 };

// CIRCUNFERENCIA

// a125e4
911 pt getcenter(pt a, pt b, pt c) { // centro da circunf
dado 3 pontos
174     b = (a + b) / 2;
2ae     c = (a + c) / 2;
98b     return inter(line(b, b + rotate90(a - b)),
3f8         line(c, c + rotate90(a - c)));
cbb }

// cd80c0
4b3 vector<pt> circ_line_inter(pt a, pt b, pt c, ld r) { //
intersecao da circunf (c, r) e reta ab
8af     vector<pt> ret;
f2b     b = b-a, a = a-c;
4b1     ld A = b*b;
20a     ld B = a*b;
2e9     ld C = a*a - r*r;
1fa     ld D = B*B - A*C;
818     if (D < -eps) return ret;
dc5     ret.push_back(c+a+b*(-B+sqrt(D+eps))/A);
20e     if (D > eps) ret.push_back(c+a+b*(-B-sqrt(D))/A);
edf     return ret;

```

```

cbb }

// fb11d8
ad2 vector<pt> circ_inter(pt a, pt b, ld r, ld R) { //
intersecao da circunf (a, r) e (b, R)
8af     vector<pt> ret;
b7e     ld d = dist(a, b);
5ce     if (d > r+R or d+min(r, R) < max(r, R)) return ret;
398     ld x = (d*d-R*R+r*r)/(2*d);
183     ld y = sqrt(r*r-x*x);
325     pt v = (b-a)/d;
76e     ret.push_back(a+v*x + rotate90(v)*y);
2cb     if (y > 0) ret.push_back(a+v*x - rotate90(v)*y);
edf     return ret;
cbb }

// 3a44fb
6e0 bool operator <(const line& a, const line& b) { //
comparador pra reta
// assume que as retas tem p < q
a13     pt v1 = a.q - a.p, v2 = b.q - b.p;
f82     if (!eq(angle(v1), angle(v2))) return angle(v1) <
angle(v2);
780     return ccw(a.p, a.q, b.p); // mesmo angulo
cbb }
b14 bool operator ==(const line& a, const line& b) {
76c     return !(a < b) and !(b < a);
cbb }

// comparador pro set pra fazer sweep line com segmentos
// 36729f
2c4 struct cmp_sweepline {
d80     bool operator () (const line& a, const line& b)
const {
// assume que os segmentos tem p < q
191         if (a.p == b.p) return ccw(a.p, a.q, b.q);
231         if (!eq(a.p.x, a.q.x) and (eq(b.p.x, b.q.x) or
a.p.x+eps < b.p.x))
780             return ccw(a.p, a.q, b.p);
dc0         return ccw(a.p, b.q, b.p);
cbb     }
214 };

```



```

// comparador pro set pra fazer sweep angle com segmentos
// f778aa
bef pt dir;
5b0 struct cmp_sweepangle {
d80     bool operator () (const line& a, const line& b)
const {
522         return get_t(dir, a) + eps < get_t(dir, b);
cbb     }
214 };

```

## 7.7 Primitivas Geometricas 3D

```

c83 typedef double ld;
e3b const ld DINF = 1e18;
107 const ld eps = 1e-9;

b32 #define sq(x) ((x)*(x))

d97 bool eq(ld a, ld b) {
ba0     return abs(a - b) <= eps;
cbb }

b2a struct pt { // ponto
2eb     ld x, y, z;
a50     pt(ld x_ = 0, ld y_ = 0, ld z_ = 0) : x(x_),
y(y_), z(z_) {}
5bc     bool operator < (const pt p) const {
059         if (!eq(x, p.x)) return x < p.x;
f98         if (!eq(y, p.y)) return y < p.y;
44c         if (!eq(z, p.z)) return z < p.z;
bb3         return 0;
cbb     }
a83     bool operator == (const pt p) const {
41c         return eq(x, p.x) and eq(y, p.y) and
eq(z, p.z);
cbb     }
44b     pt operator + (const pt p) const { return
pt(x+p.x, y+p.y, z+p.z); }
392     pt operator - (const pt p) const { return
pt(x-p.x, y-p.y, z-p.z); }
fb7     pt operator * (const ld c) const { return pt(x*c

```

```

, y*c , z*c ); }
7a1     pt operator / (const ld c) const { return pt(x/c
, y/c , z/c ); }
a65     ld operator * (const pt p) const { return x*p.x
+ y*p.y + z*p.z; }
7f6     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); }
5ed     friend istream& operator >> (istream& in, pt& p)
{
9bf         return in >> p.x >> p.y >> p.z;
cbb     }
214 };

b3a struct line { // reta
730     pt p, q;
0d6     line() {}
4b8     line(pt p_, pt q_) : p(p_), q(q_) {}
8d7     friend istream& operator >> (istream& in, line&
r) {
4cb         return in >> r.p >> r.q;
cbb     }
214 };

79b struct plane { // plano
7e1     array<pt, 3> p; // pontos que definem o plano
29b     array<ld, 4> eq; // equacao do plano
bb7     plane() {}
fb0     plane(pt p_, pt q_, pt r_) : p({p_, q_, r_}) {
build(); }

ca9     friend istream& operator >> (istream& in, plane&
p) {
2ab         return in >> P.p[0] >> P.p[1] >> P.p[2];
70e         P.build();
cbb     }
0a8     void build() {
da2         pt dir = (p[1] - p[0]) ^ (p[2] - p[0]);
7d5         eq = {dir.x, dir.y, dir.z,
dir*p[0]*(-1)};
cbb     }
214 };

```

```

// 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
2fb pt convert(ld rho, ld th, ld phi) {
cf4     return pt(sin(phi) * cos(th), sin(phi) *
sin(th), cos(phi)) * rho;
cbb }

// projecao do ponto p na reta r
256 pt proj(pt p, line r) {
bea     if (r.p == r.q) return r.p;
97a     r.q = r.q - r.p; p = p - r.p;
9f8     pt proj = r.q * ((p*r.q) / (r.q*r.q));
2cd     return proj + r.p;
cbb }

// projecao do ponto p no plano P
b1a pt proj(pt p, plane P) {
7b6     p = p - P.p[0], P.p[1] = P.p[1] - P.p[0], P.p[2]
= P.p[2] - P.p[0];
b69     pt norm = P.p[1] ^ P.p[2];
6ab     pt proj = p - (norm * (norm * p) / (norm*norm));
467     return proj + P.p[0];
cbb }

// distancia
a45 ld dist(pt a, pt b) {
fd9     return sqrt(sq(a.x-b.x) + sq(a.y-b.y) +
sq(a.z-b.z));
cbb }

// distancia ponto reta
137 ld distline(pt p, line r) {
ce1     return dist(p, proj(p, r));
cbb }

// distancia de ponto para segmento
d43 ld distseg(pt p, line r) {
73d     if ((r.q - r.p)*(p - r.p) < 0) return dist(r.p,
p);
951     if ((r.p - r.q)*(p - r.q) < 0) return dist(r.q,

```

```

p);
200     return distline(p, r);
cbb }

// distancia de ponto a plano com sinal
7cc ld sdist(pt p, plane P) {
150     return P.eq[0]*p.x + P.eq[1]*p.y + P.eq[2]*p.z +
P.eq[3];
cbb }

// distancia de ponto a plano
768 ld distplane(pt p, plane P) {
c3e     return abs(sdist(p, P));
cbb }

// se ponto pertence a reta
099 bool isinseg(pt p, line r) {
a32     return eq(distseg(p, r), 0);
cbb }

// se ponto pertence ao triangulo definido por P.p
cd2 bool isinpol(pt p, vector<pt> v) {
fad     assert(v.size() >= 3);
bf4     pt norm = (v[1]-v[0]) ^ (v[2]-v[1]);
8a4     bool inside = true;
cec     int sign = -1;
f14     for (int i = 0; i < v.size(); i++) {
834         line r(v[(i+1)%3], v[i]);
2a9         if (isinseg(p, r)) return true;

4ef         pt ar = v[(i+1)%3] - v[i];
320         if (sign == -1) sign =
((ar^(p-v[i]))*norm > 0);
82b         else if (((ar^(p-v[i]))*norm > 0) !=
sign) inside = false;
cbb     }
aca     return inside;
cbb }

// distancia de ponto ate poligono
361 ld distpol(pt p, vector<pt> v) {
3e7     pt p2 = proj(p, plane(v[0], v[1], v[2]));

```

```

61a         if (isinpola(p2, v)) return dist(p, p2);
349         ld ret = DINF;
f14         for (int i = 0; i < v.size(); i++) {
6af             int j = (i+1)%v.size();
5ee             ret = min(ret, distseg(p, line(v[i],
v[j])));
cbb         }
edf         return ret;
cbb     }

// 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
e51 enum RETCODE {BOTH, ONE, PARAL, CONCOR};
26b pair<RETCODE, pt> intersect(plane P, line r) {
fac     ld d1 = sdist(r.p, P);
f8f     ld d2 = sdist(r.q, P);
53a     if (eq(d1, 0) and eq(d2, 0))
504         return pair(BOTH, r.p);
72c     if (eq(d1, 0))
847         return pair(ONE, r.p);
485     if (eq(d2, 0))
168         return pair(ONE, r.q);
3fb     if ((d1 > 0 and d2 > 0) or (d1 < 0 and d2 < 0)) {
463         if (eq(d1-d2, 0)) return pair(PARAL, pt());
406         return pair(CONCOR, pt());
cbb     }
c84     ld frac = d1 / (d1 - d2);
3ff     pt res = r.p + ((r.q - r.p) * frac);
394     return pair(ONE, res);
cbb }

// rotaciona p ao redor do eixo u por um angulo a
787 pt rotate(pt p, pt u, ld a) {
773     u = u / dist(u, pt());
e6f     return u * (u * p) + (u ^ p ^ u) * cos(a) + (u ^
p) * sin(a);
cbb }

```

## 7.8 Primitivas Geometricas Inteiras

```

2de #define sq(x) ((x)*(ll)(x))

// 840720
b2a struct pt { // ponto
e91     int x, y;
df1     pt(int x_ = 0, int y_ = 0) : x(x_), y(y_) {}
5bc     bool operator < (const pt p) const {
95a         if (x != p.x) return x < p.x;
89c         return y < p.y;
cbb     }
a83     bool operator == (const pt p) const {
d74         return x == p.x and y == p.y;
cbb     }
cb9     pt operator + (const pt p) const { return pt(x+p.x,
y+p.y); }
a24     pt operator - (const pt p) const { return pt(x-p.x,
y-p.y); }
0ef     pt operator * (const int c) const { return pt(x*c,
y*c); }
60d     ll operator * (const pt p) const { return x*(ll)p.x
+ y*(ll)p.y; }
d86     ll operator ^ (const pt p) const { return x*(ll)p.y
- y*(ll)p.x; }
5ed     friend istream& operator >> (istream& in, pt& p) {
e37         return in >> p.x >> p.y;
cbb     }
214 };

// 7ab617
b3a struct line { // reta
730     pt p, q;
0d6     line() {}
4b8     line(pt p_, pt q_) : p(p_), q(q_) {}
8d7     friend istream& operator >> (istream& in, line& r) {
4cb         return in >> r.p >> r.q;
cbb     }
214 };

// PONTO & VETOR

```

```

// 51563e
ea8 ll dist2(pt p, pt q) { // quadrado da distancia
f24     return sq(p.x - q.x) + sq(p.y - q.y);
cbb }

// bf431d
5a2 ll sarea2(pt p, pt q, pt r) { // 2 * area com sinal
586     return (q-p)^(r-q);
cbb }

// a082d3
e32 bool col(pt p, pt q, pt r) { // se p, q e r sao colin.
034     return sarea2(p, q, r) == 0;
cbb }

// 42bb09
0cd bool ccw(pt p, pt q, pt r) { // se p, q, r sao ccw
276     return sarea2(p, q, r) > 0;
cbb }

// fcf924
c31 int quad(pt p) { // quadrante de um ponto
dbb     return (p.x<0)^3*(p.y<0);
cbb }

// 77187b
2df bool compare_angle(pt p, pt q) { // retorna se ang(p) <
ang(q)
9fc     if (quad(p) != quad(q)) return quad(p) < quad(q);
ea1     return ccw(q, pt(0, 0), p);
cbb }

// e4ad5e
ab1 pt rotate90(pt p) { // rotaciona 90 graus
a0d     return pt(-p.y, p.x);
cbb }

// RETA

// c9f07f
099 bool isinseg(pt p, line r) { // se p pertence ao seg de r
f65     pt a = r.p - p, b = r.q - p;

```

```

2ac     return (a ^ b) == 0 and (a * b) <= 0;
cbb }

// 35998c
676 bool interseg(line r, line s) { // se o seg de r
intersecta o seg de s
19b     if (isinseg(r.p, s) or isinseg(r.q, s)
c21         or isinseg(s.p, r) or isinseg(s.q, r)) return 1;

9fa     return ccw(r.p, r.q, s.p) != ccw(r.p, r.q, s.q) and
413         ccw(s.p, s.q, r.p) != ccw(s.p, s.q, r.q);
cbb }

// dd8702
9e0 int segpoints(line r) { // numero de pontos inteiros no
segmento
9ce     return 1 + __gcd(abs(r.p.x - r.q.x), abs(r.p.y -
r.q.y));
cbb }

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

// POLIGONO

// quadrado da distancia entre os retangulos a e b
(lados paralelos aos eixos)
// assume que ta representado (inferior esquerdo,
superior direito)
// e13018
485 ll dist2_rect(pair<pt, pt> a, pair<pt, pt> b) {
c59     int hor = 0, vert = 0;
34b     if (a.second.x < b.first.x) hor = b.first.x -
a.second.x;
f5f     else if (b.second.x < a.first.x) hor = a.first.x -
b.second.x;
4fd     if (a.second.y < b.first.y) vert = b.first.y -
a.second.y;
80a     else if (b.second.y < a.first.y) vert = a.first.y -

```

```

    b.second.y;
869     return sq(hor) + sq(vert);
cbb }

// d5f693
9c3 ll polarea2(vector<pt> v) { // 2 * area do poligono
b73     ll ret = 0;
c6e     for (int i = 0; i < v.size(); i++)
532         ret += sarea2(pt(0, 0), v[i], v[(i + 1) %
v.size()]);
d03     return abs(ret);
cbb }

// se o ponto ta dentro do poligono: retorna 0 se ta
fora,
// 1 se ta no interior e 2 se ta na borda
// afd587
8e7 int inpol(vector<pt>& v, pt p) { // O(n)
8de     int qt = 0;
f14     for (int i = 0; i < v.size(); i++) {
bda         if (p == v[i]) return 2;
6af         int j = (i+1)%v.size();
cc6         if (p.y == v[i].y and p.y == v[j].y) {
547             if ((v[i]-p)*(v[j]-p) <= 0) return 2;
5e2             continue;
cbb         }
78c         bool baixo = v[i].y < p.y;
057         if (baixo == (v[j].y < p.y)) continue;
366         auto t = (p-v[i])^(v[j]-v[i]);
2ad         if (!t) return 2;
0bb         if (baixo == (t > 0)) qt += baixo ? 1 : -1;
cbb     }
b84     return qt != 0;
cbb }

// 32623c
138 vector<pt> convex_hull(vector<pt> v) { // convex hull -
O(n log(n))
52d     if (v.size() <= 1) return v;
526     vector<pt> l, u;
fca     sort(v.begin(), v.end());
f14     for (int i = 0; i < v.size(); i++) {

```

```

543         while (l.size() > 1 and !ccw(l[l.size()-2],
l.back(), v[i]))
364             l.pop_back();
c35         l.push_back(v[i]);
cbb     }
3e9     for (int i = v.size() - 1; i >= 0; i--) {
2eb         while (u.size() > 1 and !ccw(u[u.size()-2],
u.back(), v[i]))
7a8             u.pop_back();
a95         u.push_back(v[i]);
cbb     }
cfc     l.pop_back(); u.pop_back();
82b     for (pt i : u) l.push_back(i);
792     return l;
cbb }

// af2d96
786 ll interior_points(vector<pt> v) { // pontos inteiros
dentro de um poligono simples
c4e     ll b = 0;
c6e     for (int i = 0; i < v.size(); i++)
0ce         b += segpoints(line(v[i], v[(i+1)%v.size()])) -
1;
a1c     return (polarea2(v) - b) / 2 + 1;
cbb }

483 struct convex_pol {
f50     vector<pt> pol;

// nao pode ter ponto colinear no convex hull
d98     convex_pol() {}
a04     convex_pol(vector<pt> v) : pol(convex_hull(v)) {}

// se o ponto ta dentro do hull - O(log(n))
// 800813
8af     bool is_inside(pt p) {
eae         if (pol.size() == 1) return p == pol[0];
67f         int l = 1, r = pol.size();
40c         while (l < r) {
ee4             int m = (l+r)/2;
48f             if (ccw(p, pol[0], pol[m])) l = m+1;
ef3             else r = m;

```

```

cbb      }
00a      if (l == 1) return isinseg(p, line(pol[0],
      pol[1]));
9e7      if (l == pol.size()) return false;
1c0      return !ccw(p, pol[l], pol[l-1]);
cbb      }
      // ponto extremo em relacao a cmp(p, q) = p mais
      // extremo q
      // (copiado de
      // https://github.com/gustavoM32/caderno-zika)
      // 56ccd2
719      int extreme(const function<bool(pt, pt)>& cmp) {
b1c          int n = pol.size();
4a2          auto extr = [&](int i, bool& cur_dir) {
22a              cur_dir = cmp(pol[(i+1)%n], pol[i]);
61a              return !cur_dir and !cmp(pol[(i+n-1)%n],
      pol[i]);
214          };
63d          bool last_dir, cur_dir;
a0d          if (extr(0, last_dir)) return 0;
993          int l = 0, r = n;
ead          while (l+1 < r) {
ee4              int m = (l+r)/2;
f29              if (extr(m, cur_dir)) return m;
44a              bool rel_dir = cmp(pol[m], pol[l]);
b18              if ((!last_dir and cur_dir) or
261                  (last_dir == cur_dir and rel_dir ==
      cur_dir)) {
8a6                  l = m;
1f1                  last_dir = cur_dir;
b6c              } else r = m;
cbb          }
792          return l;
cbb      }
316      int max_dot(pt v) {
ec1          return extreme([&](pt p, pt q) { return p*v >
      q*v; });
cbb      }
a54      pair<int, int> tangents(pt p) {
08c          auto L = [&](pt q, pt r) { return ccw(p, q, r);
      };
422          auto R = [&](pt q, pt r) { return ccw(p, r, q);

```

```

      };
fa8          return {extreme(L), extreme(R)};
cbb      }
214 };

      // dca598
6e0      bool operator <(const line& a, const line& b) { //
      comparador pra reta
      // assume que as retas tem p < q
a13          pt v1 = a.q - a.p, v2 = b.q - b.p;
036          bool b1 = compare_angle(v1, v2), b2 =
      compare_angle(v2, v1);
73c          if (b1 or b2) return b1;
780          return ccw(a.p, a.q, b.p); // mesmo angulo
cbb      }
b14      bool operator ==(const line& a, const line& b) {
76c          return !(a < b) and !(b < a);
cbb      }

      // comparador pro set pra fazer sweep line com segmentos
      // 6774df
2c4      struct cmp_sweepline {
d80          bool operator () (const line& a, const line& b)
      const {
      // assume que os segmentos tem p < q
191          if (a.p == b.p) return ccw(a.p, a.q, b.q);
614          if (a.p.x != a.q.x and (b.p.x == b.q.x or a.p.x
      < b.p.x))
780              return ccw(a.p, a.q, b.p);
dc0          return ccw(a.p, b.q, b.p);
cbb      }
214 };

      // comparador pro set pra fazer sweep angle com segmentos
      // 1ee7f5
bef      pt dir;
5b0      struct cmp_sweepangle {
d80          bool operator () (const line& a, const line& b)
      const {
261              return get_t(dir, a) < get_t(dir, b);
cbb          }
214 };

```

## 8 Extra

### 8.1 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;
}
```

### 8.2 vimrc

```
set ts=4 si ai sw=4 nu mouse=a undofile
syntax on
```

### 8.3 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();
    }
}
```

```
    }
};
```

### 8.4 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);
}
```

### 8.5 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);
}
```

### 8.6 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(...)
#endif

```

## 8.7 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

```

## 8.8 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

```

## 8.9 hash.sh

```

# Para usar (hash das linhas [11, 12]):

```

```

# bash hash.sh arquivo.cpp 11 12
sed -n $2', '$3' p' $1 | sed '/^#w/d' | cpp -dD -P
    -fpreprocessed | tr -d '[:space:]' | md5sum | cut -c-6

```

## 8.10 linehash.sh

```

# Para usar:
# bash linehash.sh arquivo.cpp
while read l; do
    echo $l > tmp.txt
    h=$(echo $(bash hash.sh tmp.txt 1 1) | cut -c-3)
    echo "$h $l"
done < "$1"

```