

Conteúdo

1 Data Structures

1.1	BIT	1
1.2	BIT2D	1
1.3	BIT2DSparse	1
1.4	PrefixSum2D	2
1.5	SegTree	2
1.6	SegTree Lazy	2
1.7	SegTree Persistente	2
1.8	SegTree Iterativa	3
1.9	SegTree Lazy Iterativa	3
1.10	SparseTable	3

2 dp

2.1	Digit DP	3
2.2	LCS	4
2.3	LIS	4
2.4	SOS DP	4

3 Geometry

3.1	ConvexHull	4
3.2	Geometry - General	4
3.3	LineContainer	5

4 Grafos

4.1	2-SAT	5
4.2	BlockCutTree	5
4.3	Centroid Decomposition	6
4.4	Dijkstra	6
4.5	Dinic	7
4.6	DSU	7
4.7	DSU Rollback	7
4.8	DSU Persistente	7
4.9	Euler Path	8
4.10	HLD	8
4.11	LCA	9
4.12	MinCostMaxFlow	9
4.13	SCC - Kosaraju	9
4.14	Tarjan	10

5 Strings

5.1	hash2	10
5.2	KMP	11
5.3	Aho-Corasick	11
5.4	Manacher	11
5.5	trie	11
5.6	Z-Function	12

6 others

6.1	Hungarian	12
6.2	MO	12
6.3	MO Tree	13

7 Math

7.1	fexp	13
7.2	CRT	13

8 Theorems

8.1	Propriedades Matemáticas	13
8.2	Geometria	14
8.3	Grafos	14
8.4	DP	15

9 Extra

9.1	Hash Function	15
-----	---------------	----

1 Data Structures

1.1 BIT

```

struct BIT {
    vector<int> bit;
    int N;

    BIT() {}
    BIT(int n) : N(n+1), bit(n+1) {}

    void update(int pos, int val) {
        for(; pos < N; pos += pos & (-pos))
            bit[pos] += val;
    }

    int query(int pos) {
        int sum = 0;
        for(; pos > 0; pos -= pos & (-pos))
            sum += bit[pos];
        return sum;
    }
};

```

1.2 BIT2D

Complexity: $O(\log^2 N)$

Bit.update(x, y, v); //Adiciona +v na posicao {x, y} da BIT

Bit.query(x, y); //Retorna o somatorio do retangulo de inicio {1, 1} e fim {x, y}

Bit.queryArea(xi, yi, xf, yf); //Retorna o somatorio do retangulo de inicio {xi, yi} e fim {xf, yf}

Bit.updateArea(xi, yi, xf, yf, v); //adiciona +v no retangulo de inicio {xi, yi} e fim {xf, yf}

IMPORTANTE! UpdateArea NAO atualiza o valor de todas as celulas no retangulo!!! Deve ser usado para Color Update

IMPORTANTE! Use query(x, y) Para acessar o valor da posicao (x, y) quando estiver usando UpdateArea

IMPORTANTE! Use queryArea(x, y, x, y) Para acessar o valor da posicao (x, y) quando estiver usando Update Padrao

```

const int MAXN = 1e3 + 5;

struct BIT2D {
    int bit[MAXN][MAXN];

    void update(int X, int Y, int val) {
        for(int x = X; x < MAXN; x += x & (-x))
            for(int y = Y; y < MAXN; y += y & (-y))
                bit[x][y] += val;
    }

    int query(int X, int Y) {
        int sum = 0;
        for(int x = X; x > 0; x -= x & (-x))
            for(int y = Y; y > 0; y -= y & (-y))
                sum += bit[x][y];
        return sum;
    }

    void updateArea(int xi, int yi, int xf, int yf, int val); //
        Same of BIT2DSparse

    int queryArea(int xi, int yi, int xf, int yf); //Same of
        BIT2DSparse
};

```

1.3 BIT2DSparse

Sparse Binary Indexed Tree 2D

Recebe o conjunto de pontos que serao usados para fazer os updates e as queries e cria uma BIT 2D esparsa que independe do "tamanho do grid".

Build: $O(N \log N)$ (N -> Quantidade de Pontos)

Query/Update: $O(\log N)$

IMPORTANTE! **Offline!**

BIT2D(pts); // pts -> vecotor<pii> com todos os pontos em que serao feitas queries ou updates

```

AA8 #define upper(v, x) (upper_bound(begin(v), end(v), x) -
begin(v))

```

```

4BA struct BIT2D {
6C1     vector<int> ord;
C03     vector<vector<int>> bit, coord;

```

```

8A4     BIT2D(vector<pii> pts) {
B03         sort(begin(pts), end(pts));

```

```

7D3         for(auto [x, y] : pts)
76B             if(ord.empty() || x != ord.back())
580                 ord.push_back(x);

```

```

261         bit.resize(ord.size() + 1);
3EB         coord.resize(ord.size() + 1);

```

```

CC7         sort(begin(pts), end(pts), [&](pii &a, pii &b) { return
a.second < b.second; });

```

```

7D3         for(auto [x, y] : pts)
837             for(int i=upper(ord, x); i < bit.size(); i += i&-i)
3E1                 if(coord[i].empty() || coord[i].back() != y)
739                     coord[i].push_back(y);

```

```

A22         for(int i=0; i<bit.size(); i++) bit[i].assign(coord[i]
].size()+1, 0);
461     }

```

```

599     void update(int X, int Y, int v) {
784         for(int i = upper(ord, X); i<bit.size(); i += i&-i)
609             for(int j = upper(coord[i], Y); j < bit[i].size(); j
+= j&-j)
9ED                 bit[i][j] += v;
7E5     }

```

```

698     int query(int X, int Y) {
A93         int sum = 0;
2C2         for(int i = upper(ord, X); i > 0; i -= i&-i)
40B             for(int j = upper(coord[i], Y); j > 0; j -= j&-j)
B03                 sum += bit[i][j];
E66         return sum;
398     }

```

```

81E     void updateArea(int xi, int yi, int xf, int yf, int val)
{
C02         update(xi, yi, val);
061         update(xf+1, yi, -val);
2ED         update(xi, yf+1, -val);
2BC         update(xf+1, yf+1, val);
49B     }

```

```
E1E   int queryArea(int xi, int yi, int xf, int yf){
ABD       return query(xf, yf) - query(xf, yi-1) - query(xi-1,
        yf) + query(xi-1, yi-1);
873   }
195   };
```

1.4 PrefixSum2D

```
const int MAXN = 1e3 + 5;
int ps [MAXN][MAXN];

void calcPS2d(){
    for (int i = 1; i < MAXN; i++) ps[0][i] += ps[0][i - 1]; //
        inicializo a la linha
    for (int i = 1; i < MAXN; i++) ps[i][0] += ps[i - 1][0]; //
        inicializo a la columna

    for (int i = 1; i < MAXN; i++)
        for (int j = 1; j < MAXN; j++)
            ps[i][j] += ps[i - 1][j] + ps[i][j - 1] - ps[i - 1][j -
                1];
}

int queryPS2d(int xi, int yi, int xf, int yf){ return ps[xf][
    yf] - ps[xf][yi-1] - ps[xi-1][yf] + ps[xi-1][yi-1]; }
```

1.5 SegTree

```
template<typename T> struct SegTree {
    vector<T> seg;
    int N;
    SegTree(int n) : N(n), seg(4*n) {}
    SegTree(vector<T> &lista) : N(lista.size()), seg(4*N) {
        build(1, 0, N-1, lista); }
    T join(T lv, T rv){ return lv + rv; }

    T query(int ls, int rs){ return query (1, 0, N-1, ls, rs); }
    void update(int pos, T v){ update(1, 0, N-1, pos, v); }

    T query(int no, int l, int r, int a, int b){
        if(b < 1 || r < a) return 0;
        if(a <= 1 && r <= b) return seg[no];
        int m=(l+r)/2, e=no*2, d=no*2+1;

        return join(query(e, l, m, a, b), query(d, m+1, r, a, b));
    }
    void update(int no, int l, int r, int pos, T v){
        if(pos < 1 || r < pos) return;
        if(l == r){ seg[no] = v; return; } // set value -> change
            to += to sum
        int m=(l+r)/2, e=no*2, d=no*2+1;

        update(e, l, m, pos, v);
        update(d, m+1, r, pos, v);

        seg[no] = join(seg[e], seg[d]);
    }
    void build(int no, int l, int r, vector<T> &lista){
        if(l == r){ seg[no] = lista[l]; return; }
        int m=(l+r)/2, e=no*2, d=no*2+1;

        build(e, l, m, lista);
        build(d, m+1, r, lista);
```

```
        seg[no] = join(seg[e], seg[d]);
    }
};

const int MAXN = 1e6 + 5;
int seg[4*MAXN];
int lazy[4*MAXN];

void unlazy(int no, int l, int r){
    if(lazy[no] == 0) return;

    int m=(l+r)/2, e=no*2, d=no*2+1;

    seg[no] += (r-l+1) * lazy[no];

    if(l != r){
        lazy[e] += lazy[no];
        lazy[d] += lazy[no];
    }

    lazy[no] = 0;
}

int query(int no, int l, int r, int a, int b){
    unlazy(no, l, r);
    if(b < 1 || r < a) return 0;
    if(a <= 1 && r <= b) return seg[no];

    int m=(l+r)/2, e=no*2, d=no*2+1;

    return query(e, l, m, a, b) + query(d, m+1, r, a, b);
}

void update(int no, int l, int r, int a, int b, int v){
    unlazy(no, l, r);
    if(b < 1 || r < a) return;
    if(a <= 1 && r <= b)
    {
        lazy[no] += v;
        unlazy(no, l, r);
        return;
    }

    int m=(l+r)/2, e=no*2, d=no*2+1;

    update(e, l, m, a, b, v);
    update(d, m+1, r, a, b, v);

    seg[no] = seg[e] + seg[d];
}

void build(int no, int l, int r, vector<int> &lista){
    if(l == r){ seg[no] = lista[l-1]; return; }

    int m=(l+r)/2, e=no*2, d=no*2+1;

    build(e, l, m, lista);
    build(d, m+1, r, lista);

    seg[no] = seg[e] + seg[d];
}
```

-> Segment Tree - Lazy Propagation com:
- Query em Range
- Update em Range

1.6 SegTree Lazy

```
build (1, 1, n, lista);
query (1, 1, n, a, b);
update(1, 1, n, a, b, x);

|   n   | o tamanho maximo da lista
| [a, b] | o intervalo da busca ou update
|   x   | o novo valor a ser somada no intervalo [a, b]
| lista | o array de elementos originais

Build: O(N)
Query: O(log N)
Update: O(log N)
Unlazy: O(1)
```

1.7 SegTree Persistente

```
-> Segment Tree Persistente: (2x mais rapido que com ponteiro)
Build(1, N) -> Cria uma Seg Tree completa de tamanho N;
    RETORNA o NodeId da Raiz
Update(Root, pos, v) -> Soma +V em POS; RETORNA o NodeId da
    nova Raiz;
Query(Root, a, b) -> RETORNA o valor do range [a, b];
Kth(RootL, RootR, K) -> Faz uma Busca Binaria na Seg de
    diferenca entre as duas versoes.
[ Root -> No Raiz da Versao da Seg na qual se quer realizar a
    operacao ]

Build: O(N) !!! Sempre chame o Build
Query: O(log N)
Update: O(log N)
Kth: O(Log N)
```

```
80E const int MAXN = 1e5 + 5;
2D8 const int MAXLOG = 31 - __builtin_clz(MAXN) + 1;
4B4 typedef int NodeId;
6E2 typedef int STp;

EA9 const STp NEUTRO = 0;
B50 int IDN, LSEG, RSEG;
519 extern struct Node NODES[];

BF2 struct Node {
AEE     STp val;
1BC     NodeId L, R;
9DA     Node(STp v = NEUTRO) : val(v), L(-1), R(-1) {}
2F4     Node& l(){ return NODES[L]; }
F2E     Node& r(){ return NODES[R]; }
5A4 };

318 Node NODES[4*MAXN + MAXLOG*MAXN]; //!!!CUIDADO COM O
    TAMANHO (aumente se necessario)
1E7 pair<Node&, NodeId> newNode(STp v = NEUTRO){ return {NODES
    [IDN] = Node(v), IDN++}; }

C3F STp join(STp lv, STp rv){ return lv + rv; }

8B5 NodeId build(int l, int r, bool root=true){
85B     if(root) LSEG = 1, RSEG = r;
844     if(l == r) return newNode().second;

EE4     int m = (l+r)/2;
DC6     auto [node, id] = newNode();

C12     node.L = build(l, m, false);
373     node.R = build(m+1, r, false);
45D     node.val = join(node.l().val, node.r().val);
```

```

648     return id;
9D5 }

2F1 NodeId update(NodeId node, int l, int r, int pos, int v){
703     if( pos < l || r < pos ) return node;
D99     if(l == r) return newNode(NODES[node].val + v).second;

EE4     int m = (l+r)/2;
BE4     auto [nw, id] =newNode();

E2C     nw.L = update(NODES[node].L, l, m, pos, v);
D4A     nw.R = update(NODES[node].R, m+1, r, pos, v);

6EC     nw.val = join(nw.l().val, nw.r().val);

648     return id;
938 }
8C0 NodeId update(NodeId node, int pos, STp v){ return update(
    node, LSEG, RSEG, pos, v); }

BFA int query(Node& node, int l, int r, int a, int b){
83C     if(b < l || r < a) return NEUTRO;
65A     if(a <= l && r <= b) return node.val;

EE4     int m = (l+r)/2;

083     return join(query(node.l(), l, m, a, b), query(node.r(),
    m+1, r, a, b));
7B5 }
8B3 int query(NodeId node, int a, int b){ return query(NODES[
    node], LSEG, RSEG, a, b); }

D0A int kth(Node& Left, Node& Right, int l, int r, int k){
3CE     if(l == r) return l;

A3B     int sum =Right.l().val - Left.l().val;
EE4     int m = (l+r)/2;

BBB     if(sum >= k) return kth(Left.l(), Right.l(), l, m, k);
5D8     return kth(Left.r(), Right.r(), m+1, r, k - sum);
9D7 }
A8D int kth(NodeId Left, NodeId Right, int k){ return kth(
    NODES[Left], NODES[Right], LSEG, RSEG, k); }

```

1.8 SegTree Iterativa

```

CD5 template<typename T> struct SegTree {
1A8     int n;
130     vector<T> seg;
F93     T join(T&l, T&r){ return l + r; }

D5D     void init(vector<T>&base){
FC7         n = base.size();
A61         seg.resize(2*n);
8DB         for(int i=0; i<n; i++) seg[i+n] = base[i];
2E1         for(int i=n-1; i>0; i--) seg[i] = join(seg[i*2], seg[i
    *2+1]);
D60     }

B7A     T query(int l, int r){ //[L, R] & [0, n-1]
966         T ans = 0; //NEUTRO //if order matters, change to
    l_ans, r_ans
706         for(l+=n, r+=n+1; l<r; l/=2, r/=2){
294             if(l&1) ans = join(ans, seg[l++]);
1EF             if(r&1) ans = join(seg[--r], ans);
E85         }

```

```

BA7     return ans;
DDF }

FB2     void update(int i, T v){ // Set Value seg[i+=n] = v //
    change to += v to sum
CBC         for(seg[i+=n] = v; i/=2;) seg[i] = join(seg[i*2], seg[
    i*2+1]);
5E8     }
DE6 };

```

1.9 SegTree Lazy Iterativa

```

CD5 template<typename T> struct SegTree {
D16     int n, h;
97F     vector<T> seg, lzy;
1DF     vector<int> sz;
F93     T join(T&l, T&r){ return l + r; }

5C7     void init(int _n){
8FD         n = _n;
704         h = 32 - __builtin_clz(n);
A61         seg.resize(2*n);
67A         lzy.resize(n);
528         sz.resize(2*n, 1);
E3F         for(int i=n-1; i; i--) sz[i] = sz[i*2] + sz[i*2+1];
D41         // for(int i=0; i<n; i++) seg[i+n] = base[i];
D41         // for(int i=n-1; i; i--) seg[i] = join(seg[i*2], seg[
    i*2+1]);
BEC     }

45B     void apply(int p, T v){
13A         seg[p] += v * sz[p];
9F8         if(p < n) lzy[p] += v;
853     }
3B4     void push(int p){
835         for(int s=h, i=p>>s; s; s--, i=p>>s)
E15             if(lzy[i] != 0) {
561                 apply(i*2, lzy[i]);
1AD                 apply(i*2+1, lzy[i]);
BC0                 lzy[i] = 0; //NEUTRO
5D8             }
848     }
F6E     void build(int p) {
5B2         for(p/=2; p; p/= 2){
F12             seg[p] = join(seg[p*2], seg[p*2+1]);
C3C             if(lzy[p] != 0) seg[p] += lzy[p] * sz[p];
D65         }
972     }

B7A     T query(int l, int r){ //[L, R] & [0, n-1]
0ED         l+=n, r+=n+1;
F4B         push(l); push(r-1);

966         T ans = 0; //NEUTRO
DC6         for(; l<r; l/=2, r/=2){
286             if(l&1) ans = join(seg[l++], ans);
06E             if(r&1) ans = join(ans, seg[--r]);
A9D         }
BA7         return ans;
D71     }

FAB     void update(int l, int r, T v){
0ED         l+=n, r+=n+1;
F4B         push(l); push(r-1);

98D         int l0 = l, r0 = r;
DC6         for(; l<r; l/=2, r/=2){
5D1             if(l&1) apply(l++, v);

```

```

E94         if(r&1) apply(--r, v);
55B     }
FE7     build(l0); build(r0-1);
E29     }
0E4 };

1.10 SparseTable

80E const int MAXN = 1e5 + 5;
F44 const int MAXLG = 31 - __builtin_clz(MAXN) + 1;

03B int table[MAXLG][MAXN];

EDC void build(vector<int> &v){
136     int N = v.size();
606     for(int i=0; i<N; i++) table[0][i] = v[i];

671     for(int p=1; p < MAXLG; p++)
13B         for(int i=0; i + (1 << p) <= N; i++)
67C             table[p][i] = min(table[p-1][i], table[p-1][i+(1 <<
    (p-1))]);
6CD }

9E3 int query(int l, int r){
796     int p = 31 - __builtin_clz(r - l + 1); //floor log
E56     return min(table[p][l], table[p][r - (1<<p) + 1 ]);
991 }

819 Sparse Table for Range Minimum Query [L, R] [0, N)
DA9 build: O(N log N)
0EB Query: O(1)
B4F Value -> Original Array

```

2 dp

2.1 Digit DP

Digit DP - Sum of Digits

Solve(K) -> Retorna a soma dos digitos de todo numero X tal que: $0 \leq X \leq K$

dp[D][S][f] -> D: Quantidade de digitos; S: Soma dos digitos; f: Flag que indica o limite.

int limite[D] -> Guarda os digitos de K.

Complexity: $O(D^2 * B^2)$ (B = Base = 10)

```

11 dp[2][19][170];

int limite[19];
11 digitDP(int idx, int sum, bool flag){
    if(idx < 0) return sum;
    if(~dp[flag][idx][sum]) return dp[flag][idx][sum];

    dp[flag][idx][sum] = 0;
    int lm = flag ? limite[idx] : 9;

    for(int i=0; i<=lm; i++)
        dp[flag][idx][sum] += digitDP(idx-1, sum+i, (flag && i
            == lm));

    return dp[flag][idx][sum];
}

```

```
11 solve(11 k){
    memset(dp, -1, sizeof dp);

    int sz=0;
    while(k){
        limite[sz++] = k % 10LL;
        k /= 10LL;
    }

    return digitDP(sz-1, 0, true);
}
```

2.2 LCS

LCS - Longest Common Subsequence

Complexity: $O(N^2)$

* Recursive: `memset(memo, -1, sizeof memo); LCS(0, 0);`
* Iterative: `LCS_It();`

* RecoverLCS $O(N)$
Recover just one of all the possible LCS

```
const int MAXN = 5*1e3 + 5;
int memo[MAXN][MAXN];

string s, t;

inline int LCS(int i, int j){
    if(i == s.size() || j == t.size()) return 0;
    if(memo[i][j] != -1) return memo[i][j];

    if(s[i] == t[j]) return memo[i][j] = 1 + LCS(i+1, j+1);

    return memo[i][j] = max(LCS(i+1, j), LCS(i, j+1));
}

int LCS_It(){
    for(int i=s.size()-1; i>=0; i--)
        for(int j=t.size()-1; j>=0; j--)
            if(s[i] == t[j])
                memo[i][j] = 1 + memo[i+1][j+1];
            else
                memo[i][j] = max( memo[i+1][j], memo[i][j+1] );

    return memo[0][0];
}

string RecoverLCS(int i, int j){
    if(i == s.size() || j == t.size()) return "";

    if(s[i] == t[j]) return s[i] + RecoverLCS(i+1, j+1);

    if(memo[i+1][j] > memo[i][j+1]) return RecoverLCS(i+1, j);

    return RecoverLCS(i, j+1);
}
```

2.3 LIS

LIS - Longest Increasing Subsequence

Complexity: $O(N \log N)$
* For ICREASING sequence, use `lower_bound()`
* For NON DECREASING sequence, use `upper_bound()`

```
int LIS(vector<int>& nums){
    vector<int> lis;

    for(auto x : nums)
    {
        auto it = lower_bound(lis.begin(), lis.end(), x);

        if(it == lis.end()) lis.push_back(x);
        else *it = x;
    }

    return (int) lis.size();
}
```

2.4 SOS DP

SOS DP - Sum over Subsets

Dado que cada mask possui um valor inicial (iVal), computa para cada mask a soma dos valores de todas as suas submasks.

N -> Numero Maximo de Bits
iVal[mask] -> initial Value / Valor Inicial da Mask
dp[mask] -> Soma de todos os SubSets

Iterar por todas as submasks: `for(int sub=mask; sub>0; sub=(sub-1)&mask)`

```
const int N = 20;
11 dp[1<<N], iVal[1<<N];

void sosDP(){ // O(N * 2^N)
    for(int i=0; i<(1<<N); i++)
        dp[i] = iVal[i];

    for(int i=0; i<N; i++)
        for(int mask=0; mask<(1<<N); mask++)
            if(mask&(1<<i))
                dp[mask] += dp[mask^(1<<i)];
}

void sosDPsub(){ // O(3^N) //suboptimal
    for (int mask = 0, i; mask < (1<<N); mask++)
        for(i = mask, dp[mask] = iVal[0]; i>0; i=(i-1) & mask) //
            iterate over all submasks
                dp[mask] += iVal[i];
}
```

3 Geometry

3.1 ConvexHull

```
C19 struct PT {
0BE     ll x, y;
0A5     PT(ll x=0, ll y=0) : x(x), y(y) {}

0DC     PT operator- (const PT&a) const{ return PT(x-a.x, y-a.y)
; }
A68     ll operator% (const PT&a) const{ return (x*a.y - y*a.x)
; } //Cross // Vector product

5C7     bool operator==(const PT&a) const{ return x == a.x && y
== a.y; }
B4F     bool operator< (const PT&a) const{ return x != a.x ? x <
a.x : y < a.y; }
2EC };

D41 // Colinear? Mude >= 0 para > 0 nos while
CD7 vector<PT> ConvexHull(vector<PT> pts, bool sorted=false) {
EC1     if(!sorted) sort(begin(pts), end(pts));
6E7     pts.resize(unique(begin(pts), end(pts)) - begin(pts));
64B     if(pts.size() <= 1) return pts;

B4E     int s=0, n=pts.size();
988     vector<PT> h (2*n+1);

AA9     for(int i=0; i<n; h[s++] = pts[i++])
316         while(s > 1 && (pts[i] - h[s-2]) % (h[s-1] - h[s-2])
>= 0 )
351             s--;

61B     for(int i=n-2, t=s; ~i; h[s++] = pts[i--])
644         while(s > t && (pts[i] - h[s-2]) % (h[s-1] - h[s-2])
>= 0 )
351             s--;

CBB     h.resize(s-1);
81C     return h;
216 }
D41 // FOR DOUBLE POINT //
D4E See Geometry - General
```

3.2 Geometry - General

```
D40 #define ld long double

D41 // !!! NOT TESTED !!! //

C19 struct PT {
0BE     ll x, y;
0A5     PT(ll x=0, ll y=0) : x(x), y(y) {}

006     PT operator+ (const PT&a) const{ return PT(x+a.x, y+a.y)
; }
0DC     PT operator- (const PT&a) const{ return PT(x-a.x, y-a.y)
; }
954     ll operator* (const PT&a) const{ return (x*a.x + y*a.y)
; } //DOT product // norm // lenght^2 // inner
A68     ll operator% (const PT&a) const{ return (x*a.y - y*a.x)
; } //Cross // Vector product
B54     PT operator* (ll c) const{ return PT(x*c, y*c); }
B25     PT operator/ (ll c) const{ return PT(x/c, y/c); }

5C7     bool operator==(const PT&a) const{ return x == a.x && y
== a.y; }
B4F     bool operator< (const PT&a) const{ return x != a.x ? x <
a.x : y < a.y; }
F71     bool operator<<(const PT&a) const{ PT p=*this; return (p
%a == 0) ? (p*p < a*a) : (p%a < 0); } //angle(p) < angle(
a)
```

```

FD8 } ;

D41 // FOR DOUBLE POINT //
D39 const ld EPS = 1e-9;
5B4 bool eq(ld a, ld b){ return abs(a-b) < EPS; } // ==
C1E bool lt(ld a, ld b){ return a + EPS < b; } // <
D22 bool gt(ld a, ld b){ return a > b + EPS; } // >
A82 bool le(ld a, ld b){ return a < b + EPS; } // <=
410 bool ge(ld a, ld b){ return a + EPS > b; } // >=
3AE bool operator==(const PT&a) const{ return eq(x, a.x) && eq
(y, a.y); } // for double point
5EF bool operator< (const PT&a) const{ return eq(x, a.x) ? lt(
y, a.y) : lt(x, a.x); } // for double point
DBA bool operator<<(PT&a){ PT&p=*this; return eq(p%a, 0) ? lt(
p*p, a*a) : lt(p%a, 0); } //angle(this) < angle(a)
D41 //Change LL to LD and uncomment this
D41 //Also, consider replacing comparisons with these
functions

7C9 ld dist (PT a, PT b){ return sqrtl((a-b)*(a-b)); }
// distance from A to B
C43 ld angle (PT a, PT b){ return acos((a*b) / sqrtl(a*a) /
sqrtl(b*b)); } //Angle between A and B
CBB PT rotate(PT p, double ang){ return PT(p.x*cos(ang) - p.y*
sin(ang), p.x*sin(ang) + p.y*cos(ang)); } //Left rotation.
Angle in radian

EA1 ll Area(vector<PT>& p){
604 ll area = 0;
37F for(int i=2; i < p.size(); i++)
20A area += (p[i]-p[0]) % (p[i-1]-p[0]);
5BF return abs(area) / 2LL;
75B }

7EF PT intersect(PT a1, PT d1, PT a2, PT d2){
EB3 return a1 + d1 * (((a2 - a1)%d2) / (d1%d2));
14D }

9DD ld dist_pt_line(PT a, PT l1, PT l2){
E5A return abs( ((a-l1) % (l2-l1)) / dist(l1, l2) );
96D }

7EB ld dist_pt_segm(PT a, PT s1, PT s2){
E63 if(s1 == s2) return dist(s1, a);

348 PT d = s2 - s1;
9C4 ld t = max(0.0L, min(1.0L, ((a-s1)*d) / sqrtl(d*d) ));

1E8 return dist(a, s1+(d*t));
4CE }

```

3.3 LineContainer

```

72C struct Line {
3E2 mutable ll k, m, p;
CA5 bool operator<(const Line& o) const { return k < o.k; }
ABF bool operator<(ll x) const { return p < x; }
7E3 };

781 struct LineContainer : multiset<Line, less<>> {
FD2 static const ll inf = LLONG_MAX; // Double: inf = 1/.0,
div(a,b) = a/b
10F ll div(ll a, ll b) { return a / b - ((a ^ b) < 0 && a %
b); } //floored division

A1C bool isect(iterator x, iterator y) {
A95 if(y == end()) return x->p = inf, 0;
9CB if(x->k == y->k) x->p = x->m > y->m ? inf : -inf;

```

```

591 else x->p = div(y->m - x->m, x->k - y->k);
870 return x->p >= y->p;
2FA }

141 void add_line(ll k, ll m){ // kx + m //if minimum k
*=-1, m*=-1, query*=-1
116 auto z = insert({k, m, 0}), y = z++, x = y;
7B1 while(isect(y, z)) z = erase(z);
141 if(x != begin() && isect(--x, y)) isect(x, y = erase(y
));
1A4 while((y = x) != begin() && (--x)->p >= y->p) isect(x,
erase(y));
17C }

4AD ll query(ll x) {
229 assert(!empty());
7D1 auto l = *lower_bound(x);
96A return l.k * x + l.m;
D21 }
0B9 };

```

4 Grafos

4.1 2-SAT

```

2 SAT - Two Satisfiability Problem

IMPORTANTE! o grafo deve estar 0-indexado!
Retorna uma valoracao verdadeira se possivel ou um vetor
vazio se impossivel;
inverso de u = ~u

D9D struct TwoSat {
060 int N;
67E vector<vector<int>> E;

662 TwoSat(int N) : N(N), E(2 * N) {}
3E1 inline int eval(int u) const{ return u < 0 ? ((~u)+N)
%(2*N) : u; }

B0E void add_or(int u, int v){
245 E[eval(~u)].push_back(eval(v));
F37 E[eval(~v)].push_back(eval(u));
30A }
4B9 void add_nand(int u, int v) {
9FA E[eval(u)].push_back(eval(~v));
CED E[eval(v)].push_back(eval(~u));
D1C }
CEB void set_true (int u){ E[eval(~u)].push_back(eval(u)); }
5A5 void set_false(int u){ set_true(~u); }
286 void add_imply(int u, int v){ E[eval(u)].push_back(eval(
v)); }
E81 void add_and (int u, int v){ set_true(u); set_true(v);
}
347 void add_nor (int u, int v){ add_and(~u, ~v); }
A32 void add_xor (int u, int v){ add_or(u, v); add_nand(u,
v); }
F65 void add_xnor (int u, int v){ add_xor(u, ~v); }

28E vector<bool> solve() {
F18 vector<bool> ans(N);
F40 auto scc = tarjan();

51F for (int u = 0; u < N; u++)
FC2 if(scc[u] == scc[u+N]) return {}; //false

```

```

951 else ans[u] = scc[u+N] > scc[u];

BA7 return ans; //true
166 }
BF2 private:
401 vector<int> tarjan() {
C23 vector<int> low(2*N), pre(2*N, -1), scc(2*N, -1);
7B4 stack<int> st;
226 int clk = 0, ncomps = 0;

3C1 auto dfs = [&](auto&& dfs, int u) -> void {
FD2 pre[u] = low[u] = clk++;
4A6 st.push(u);

7F2 for(auto v : E[u])
325 if(pre[v] == -1) dfs(dfs, v), low[u] = min(low[u],
low[v]);
295 else
16E if(scc[v] == -1) low[u] = min(low[u], pre[v]);

8AD if(low[u] == pre[u]){
78B int v = -1;
696 while(v != u) scc[v = st.top()] = ncomps, st.pop()
;
9DF ncomps++;
CBB }
860 };

438 for(int u=0; u < 2*N; u++)
DC6 if(pre[u] == -1)
22C dfs(dfs, u);

9AB return scc; //tarjan SCCs order is the reverse of
topoSort, so (u->v if scc[v] <= scc[u])
98F }
DC3 };

```

4.2 BlockCutTree

```

Block Cut Tree - BiConnected Component

reset(n);
addEdge(u, v);
tarjan(Root);
buildBCC(n);

No fim o grafo da Block Cut Tree estara em _vector<int> tree
[]_

229 const int MAXN = 1e6 + 5;
8FA const int MAXM = 1e6 + 5; //Cuidado

7F4 vector<pii> grafo [MAXN];
C71 int pre[MAXN], low[MAXN], clk=0, C=0;

C49 vector<pii> edge;
FA4 bool visEdge[MAXN];
C7C int edgeComponent [MAXM];
316 int vertexComponent [MAXN];

B5A int cut [MAXN];
4CE stack<int> s;

20E vector<int> tree [2*MAXN];
AB3 int componentSize [2*MAXN]; //vertex - cutPoints

```

```

A20 void reset(int n){
F35   for(int i=0; i<=edge.size(); i++)
3F5     visEdge[i] = edgeComponent[i] = 0;

CD2   edge.clear();

CCD   for(int i=0; i<=n; i++){
16A     pre[i] = low[i] = -1;
B40     cut[i] = false;
DC6     vertexComponent[i] = 0;
018     grafo[i].clear();
B05   }

C5E   for(int i=0; i<=C; i++){
B2E     componentSize[i] = 0;
50A     tree[i].clear();
497   }

AA0   while(!s.empty()) s.pop();

057   clk = C = 0;
54B }

EE7 void newComponent(int i){
C43   C++;
B14   int j;

016   do {
EFE     j = s.top(); s.pop();
A54     edgeComponent[j] = C;

D69     auto [u, v] = edge[j];
68B     if(!cut[u] && !vertexComponent[u]) componentSize[C]++,
vertexComponent[u] = C;
C47     if(!cut[v] && !vertexComponent[v]) componentSize[C]++,
vertexComponent[v] = C;

80A   } while(!s.empty() && j != i);
EB5 }

C8C void tarjan(int u, bool root = true){
FD2   pre[u] = low[u] = clk++;

B10   bool any = false;
378   int chd = 0;

622   for(auto [v, i] : grafo[u]){
9CC     if(visEdge[i]) continue;
B82     visEdge[i] = true;

826     s.emplace(i);

9BE     if(pre[v] == -1)
F95     {
692       tarjan(v, false);

E7F     low[u] = min(low[v], low[u]);
87D     chd++;

68E     if(!root && low[v] >= pre[u]) cut[u] = true,
newComponent(i);
D4B     if( root && chd >= 2)         cut[u] = true,
newComponent(i);
96C     }
295     else
201     low[u] = min(low[u], pre[v]);
C2D   }

F05   if(root) newComponent(-1);

```

```

05F }

D41 //ATENCAO: ESTA 1-INDEXADO
4D6 void buildBCC(int n){
146   vector<bool> marc(C+1, false);

05A   for(int u=1; u<=n; u++)
F95   {
BE5     if(!cut[u]) continue;

C43     C++;
A24     cut[u] = C;

DC6     for(auto [v, i] : grafo[u])
F95     {
D5C       int ec = edgeComponent[i];
28B       if(!marc[ec])
F95       {
D8E         marc[ec] = true;
415         tree[cut[u]].emplace_back(ec);
D58         tree[ec].emplace_back(cut[u]);
00F       }
08F     }

DC6     for(auto [v, i] : grafo[u])
39F     marc[edgeComponent[i]] = false;
001   }
AFD }

FAE void addEdge(int u, int v){
AC3   int i = edge.size();
1C9   grafo[u].emplace_back(v, i);
179   grafo[v].emplace_back(u, i);
3B1   edge.emplace_back(u, v);
4F4 }

```

4.3 Centroid Decomposition

Centroid Decomposition

Complexity: $O(N \cdot \log N)$

dfsc() -> para criar a centroid tree

rem[u] -> True se U ja foi removido (pra dfsc)
szt[u] -> Size da subarvore de U (pra dfsc)
parent[u] -> Pai de U na centroid tree *parent[ROOT] = -1
distToAncestor[u][i] -> Distancia na arvore original de u para seu i-esimo pai na centroid tree *distToAncestor[u][0] = 0

dfsc(u=node, p=parent(subtree), f=parent(centroid tree),
sz=size of tree)

```

229 const int MAXN = 1e6 + 5;

A34 vector<int> grafo[MAXN];
BE9 deque<int> distToAncestor[MAXN];

C76 bool rem[MAXN];
BBD int szt[MAXN], parent[MAXN];

1B0 void getDist(int u, int p, int d=0){
F3E   for(auto v : grafo[u])
A6B     if(v != p && !rem[v])
334     getDist(v, u, d+1);

```

```

F0D   distToAncestor[u].emplace_front(d);
C46 }

3A5 int getSz(int u, int p){
030   szt[u] = 1;
F3E   for(auto v : grafo[u])
A6B     if(v != p && !rem[v])
35F     szt[u] += getSz(v, u);
865   return szt[u];
FD9 }

994 void dfsc(int u=0, int p=-1, int f=-1, int sz=-1){
C0F   if(sz < 0) sz = getSz(u, -1); //starting new tree

F3E   for(auto v : grafo[u])
E5C     if(v != p && !rem[v] && szt[v]*2 >= sz)
6F7     return dfsc(v, u, f, sz);

2EA   rem[u] = true, parent[u] = f;
C5E   getDist(u, -1, 0); //get subtree dists to centroid

F3E   for(auto v : grafo[u])
D8A     if(!rem[v])
D8F     dfsc(v, u, u, -1);
B0F }

```

4.4 Dijkstra

```

const int MAXN = 1e6 + 5;
#define INF 0x3f3f3f3f
#define vi vector<int>

```

```
vector<pii> grafo [MAXN];
```

```

vi dijkstra(int s){
vi dist (MAXN, INF); // !!! Change MAXN to N

priority_queue<pii, vector<pii>, greater<pii>> fil;
fila.push({0, s});
dist[s] = 0;

while(!fila.empty())
{
auto [d, u] = fila.top();
fila.pop();

if(d > dist[u]) continue;

for(auto [v, c] : grafo[u])
if( dist[v] > dist[u] + c )
{
dist[v] = dist[u] + c;
fila.push({dist[v], v});
}
}

return dist;
}

Dijkstra - Shortest Paths from Source

```

caminho minimo de um vertice u para todos os outros vertices de um grafo ponderado

Complexity: $O(N \log N)$

```
dijkstra(s)      -> s : Source, Origem. As distancias serao
                   calculadas com base no vertice s
grafo[u] = {v, c}; -> u : Vertice inicial, v : Vertice
                   final, c : Custo da aresta
priority_queue<pii, vector<pii>, greater<pii>> -> Ordena pelo
menor custo -> {d, v} -> d : Distancia, v : Vertice
```

4.5 Dinic

Dinic - Max Flow Min Cut
Algoritmo de Dinitz para encontrar o Fluxo Maximo.

Casos de Uso em [Theorems/Flow]

IMPORTANTE! O algoritmo esta 0-indexado

Complexity:

$O(V^2 * E)$ -> caso geral
 $O(\sqrt{V} * E)$ -> grafos com $cap = 1$ para toda aresta //
matching bipartido

* Informacoes:

Crie o Dinic: Dinic dinic(n, source, sink);
Adicione as Arestas: dinic.addAresta(u, v, capacity);
Para calcular o Fluxo Maximo: dinic.maxFlow()
Para saber se um vertice U esta no Corte Minimo:
dinic.inCut(u)

* Sobre o Codigo:

vector<Aresta> arestas; -> Guarda todas as arestas do
grafo e do grafo residual
vector<vector<int>> adj; -> Guarda em adj[u] os indices de
todas as arestas saindo de u
vector<int> ptr; -> Pointer para a proxima aresta ainda
nao visitada de cada vertice
vector<int> level; -> Distancia em vertices a partir do
Source. Se igual a N o vertice nao foi visitado.
A BFS retorna se Sink e alcancavel de Source. Se nao e
porque foi atingido o Fluxo Maximo
A DFS retorna um possivel aumento do Fluxo

```
7C9 struct Aresta {
37D   int u, v; ll cap;
8A7   Aresta(int u, int v, ll cap) : u(u), v(v), cap(cap) {}
475 };
```

```
14D struct Dinic {
```

```
6B0   int n, source, sink;
903   vector<vector<int>> adj;
B83   vector<Aresta> arestas;
5A0   vector<int> level, ptr; //pointer para a proxima aresta
nao saturada de cada vertice
```

```
6C1   Dinic(int n, int source, int sink) : n(n), source(source)
), sink(sink) { adj.resize(n); }
```

```
840   void addAresta(int u, int v, ll cap)
F95   {
12F       adj[u].push_back(arestas.size());
BB2       arestas.emplace_back(u, v, cap);
```

```
324       adj[v].push_back(arestas.size());
C78       arestas.emplace_back(v, u, 0);
91D   }
```

```
AD2   ll dfs(int u, ll flow = 1e9){
87D       if(flow == 0) return 0;
B2A       if(u == sink) return flow;
```

```
AD2   for(int &i = ptr[u]; i < adj[u].size(); i++)
F95   {
393       int atual = adj[u][i];
FD4       int v = arestas[atual].v;
```

```
B58       if(level[u] + 1 != level[v]) continue;
```

```
A35       if(ll got = dfs(v, min(flow, arestas[atual].cap)) )
F95       {
D82           arestas[atual].cap -= got;
A68           arestas[atual^1].cap += got;
529           return got;
FBF       }
E2C   }
```

```
BB3       return 0;
170   }
```

```
838 bool bfs(){
A9A   level = vector<int> (n, n);
51C   level[source] = 0;
```

```
A69   queue<int> fila;
B82   fila.push(source);
```

```
649   while(!fila.empty())
F95   {
ECD       int u = fila.front();
8EE       fila.pop();
```

```
E20       for(auto i : adj[u]){
CAA           int v = arestas[i].v;
```

```
269           if(arestas[i].cap == 0 || level[v] <= level[u] + 1
) continue;
```

```
789           level[v] = level[u] + 1;
2B0           fila.push(v);
602       }
822   }
```

```
348       return level[sink] < n;
6ED   }
```

```
4D1 bool inCut(int u){ return level[u] < n; }
```

```
FE4 ll maxFlow(){
04B   ll ans = 0;
```

```
6D4   while( bfs() ){
11B       ptr = vector<int> (n+1, 0);
```

```
BDD       while(ll got = dfs(source)) ans += got;
97B   }
```

```
BA7       return ans;
A65   }
E38   };
```

4.6 DSU

```
struct DSU {
    vector<int> pai, sz;
    DSU(int n) : pai(n+1), sz(n+1, 1) {
        for(int i=0; i<=n; i++) pai[i] = i;
```

```
    }

    int find(int u){ return pai[u] == u ? u : pai[u] = find(pai[
u]); }

    void join(int u, int v){
        u = find(u), v = find(v);

        if(u == v) return;
        if(sz[v] > sz[u]) swap(u, v);

        pai[v] = u;
        sz[u] += sz[v];
    }
};
```

4.7 DSU Rollback

Disjoint Set Union with **Rollback** - $O(\log n)$
checkpoint() -> salva o estado atual
rollback() -> restaura no ultimo checkpoint
save another var? +save in join & +line in pop

```
4EA struct DSUr {
ECD   vector<int> pai, sz, savept;
D35   stack<pair<int&, int>> st;
EB0   DSUr(int n) : pai(n+1), sz(n+1, 1) {
51E       for(int i=0; i<=n; i++) pai[i] = i;
6CE   }
```

```
43F   int find(int u){ return pai[u] == u ? u : find(pai[u]);
}
```

```
AF9   void join(int u, int v){
B80       u = find(u), v = find(v);
```

```
360       if(u == v) return;
844       if(sz[v] > sz[u]) swap(u, v);
```

```
A60       save(pai[v]); pai[v] = u;
5DA       save(sz[u]); sz[u] += sz[v];
047   }
```

```
2D0   void save(int &x){ st.emplace(x, x); }
42D   void pop(){
6A1       st.top().first = st.top().second; st.pop();
6A1       st.top().first = st.top().second; st.pop();
4DD   }
```

```
6E6   void checkpoint(){ savept.push_back(st.size()); }
5CF   void rollback(){
8EB       while(st.size() > savept.back()) pop();
520       savept.pop_back();
BB2   }
9E2   };
```

4.8 DSU Persistente

SemiPersistent Disjoint Set Union - $O(\log n)$
find(u, q) -> Retorna o pai de U no tempo q
* tim -> tempo em que o pai de U foi alterado


```

2CE struct DSUp {
AE4     vector<int> pai, sz, tim;
258     int t=1;
910     DSUp(int n) : pai(n+1), sz(n+1, 1), tim(n+1) {
51E         for(int i=0; i<=n; i++) pai[i] = i;
50F     }

7F9     int find(int u, int q = INT_MAX){
568         if( pai[u] == u || q < tim[u] ) return u;
8B3         return find(pai[u], q);
0A1     }

AF9     void join(int u, int v){
B80         u = find(u), v = find(v);

360         if(u == v) return;
844         if(sz[v] > sz[u]) swap(u, v);

555         pai[v] = u;
36E         tim[v] = t++;
CC3         sz[u] += sz[v];
8D8     }
96D };

```

4.9 Euler Path

Euler Path - Algoritmo de Hierholzer para caminho Euleriano

Complexity: $O(V + E)$

IMPORTANTE! O algoritmo esta 0-indexado

* Informacoes
 addEdge(u, v) -> Adiciona uma aresta de U para V
 EulerPath(n) -> Retorna o Euler Path, ou um vetor vazio se impossivel
 vi path -> vertices do Euler Path na ordem
 vi pathId -> id das Arestas do Euler Path na ordem

Euler em Undirected graph:

- Cada vertice tem um numero par de arestas (circuito); OU
- Exatamente dois vertices tem um numero impar de arestas (caminho);

Euler em Directed graph:

- Cada vertice tem quantidade de arestas |entrada| == |saida| (circuito); OU
- Exatamente 1 tem |entrada|+1 == |saida| && exatamente 1 tem |entrada| == |saida|+1 (caminho);

* Circuito -> U e o primeiro e ultimo

* Caminho -> U e o primeiro e V o ultimo

```

0C1 #define vi vector<int>

229 const int MAXN = 1e6 + 5;
210 const bool BIDIRECIONAL = true;

```

```

7F4 vector<pii> grafo[MAXN];
CBD vector<bool> used;

```

```

FAE void addEdge(int u, int v){
FD8     grafo[u].emplace_back(v, used.size()); if(BIDIRECIONAL
    && u != v)
7F0     grafo[v].emplace_back(u, used.size());
EDA     used.emplace_back(false);
3C1 }

```

```

EFB pair<vi, vi> EulerPath(int n, int src=0){
79C     int s=-1, t=-1;
E4D     vector<int> selfLoop(n*BIDIRECIONAL, 0);

C30     if(BIDIRECIONAL)
F95     {
C0F         for(int u=0; u<n; u++) for(auto&[v, id] : grafo[u]) if
(u==v) selfLoop[u]++;
19E         for(int u=0; u<n; u++)
D2B             if((grafo[u].size() - selfLoop[u])%2)
A4F                 if(t != -1) return {vi(), vi()}; // mais que 2
com grau impar
F8A                 else t = s, s = u;

C0E         if(t == -1 && t != s) return {vi(), vi()}; // so 1 com
grau impar
E78         if(s == -1 || t == src) s = src; // se
possivel, seta start como src
E07     }
295     else
F95     {
8E2         vector<int> in(n, 0), out(n, 0);

19E         for(int u=0; u<n; u++)
0DB             for(auto [v, edg] : grafo[u])
8C0                 in[v]++, out[u]++;

19E         for(int u=0; u<n; u++)
074             if(in[u] - out[u] == -1 && s == -1) s = u; else
3C0             if(in[u] - out[u] == 1 && t == -1) t = u; else
825             if(in[u] !=out[u]) return {vi(), vi()};

755         if(s == -1 && t == -1) s = t = src; // se
possivel, seta s como src
A6E         if(s == -1 && t != -1) return {vi(), vi()}; // Existe
S mas nao T
1E2         if(s != -1 && t == -1) return {vi(), vi()}; // Existe
T mas nao S
667     }

84C     for(int i=0; grafo[s].empty() && i<n; i++) s=(s+1)%n;
//evita s ser vertice isolado

D41     /////// DFS ///////
66A     vector<int> path, pathId, idx(n, 0);
982     stack<pii> st; // {Vertex, EdgeId}
D1E     st.push({s, -1});

2C8     while(!st.empty())
F95     {
723         auto [u, edg] = st.top();
B38         while(idx[u] < grafo[u].size() && used[grafo[u][idx[u]
]].second)) idx[u]++;

704         if(idx[u] < grafo[u].size())
F95         {
CAD             auto [v, id] = grafo[u][idx[u]];
3C1             used[id] = true;
F26             st.push({v, id});
5E2             continue;
2A1         }

960         path.push_back(u);
E1A         pathId.push_back(edg);
25A         st.pop();
5E9     }

301     pathId.pop_back();
023     reverse(begin(path), end(path));

```

```

6FF     reverse(begin(pathId), end(pathId));

D41     /// Grafo conexo ? ///
ADC     int edgesTotal = 0;
4B4     for(int u=0; u<n; u++) edgesTotal += grafo[u].size() + (
BIDIRECIONAL ? selfLoop[u] : 0);
0A8     if(BIDIRECIONAL) edgesTotal /= 2;
934     if(pathId.size() != edgesTotal) return {vi(), vi()};

438     return {path, pathId};
722 }

```

4.10 HLD

Heavy-Light Decomposition

Complexity: $O(\text{LogN} * (\text{qry} || \text{updt}))$

Change qry(l, r) and updt(l, r) to call a query and update structure of your will

```

HLD hld(n); //call init
hld.add_edges(u, v); //add all edges
hld.build(); //Build everthing for HLD

```

tin[u] -> Pos in the structure (Seg, Bit, ...)
 nxt[u] -> Head/Endpoint
 IMPORTANTE! o grafo deve estar 0-indexado!

```

EAA const bool EDGE = false;
403 struct HLD {
673 public:
789     vector<vector<int>>> g; //grafo
575     vector<int> sz, parent, tin, nxt;
1B1     HLD(){}
90C     HLD(int n){ init(n); }
940     void init(int n){
A34         t = 0;
8F5         g.resize(n); tin.resize(n);
7BA         sz.resize(n);nxt.resize(n);
62B         parent.resize(n);
D94     }
FAE     void addEdge(int u, int v){
7EA         g[u].emplace_back(v);
4A3         g[v].emplace_back(u);
1DB     }
1F8     void build(int root=0){
E4A         nxt[root]=root;
043         dfs(root, root);
7D9         hld(root, root);
F40     }

3D1     ll query_path(int u, int v){
0E8         if(tin[u] < tin[v]) swap(u, v);
D63         if(nxt[u] == nxt[v]) return qry(tin[v]+EDGE, tin[u]);
7C8         return qry(tin[nxt[u]], tin[u]) + query_path(parent[
nxt[u]], v);
C6B     }

2F3     void update_path(int u, int v, ll x){
0E8         if(tin[u] < tin[v]) swap(u, v);
D55         if(nxt[u] == nxt[v]) return updt(tin[v]+EDGE, tin[u],
x);
0A7         updt(tin[nxt[u]], tin[u], x); update_path(parent[nxt[
u]], v, x);

```



```

177 }

BF2 private:
EBB ll qry(int l, int r){ if(EDGE && l>r) return 0; /*NEUTRO
*/ } //call Seg, BIT, etc
6D9 void updt(int l, int r, ll x){ if(EDGE && l>r) return; }
//call Seg, BIT, etc

FB6 void dfs(int u, int p){
573 sz[u] = 1, parent[u] = p;
E69 for(auto &v : g[u]) if(v != p) {
1FB dfs(v, u); sz[u] += sz[v];

14A if(sz[v] > sz[g[u][0]] || g[u][0] == p)
06F swap(v, g[u][0]);
7E2 }
53F }

6BB int t=0;
11E void hld(int u, int p){
2C6 tin[u] = t++;
BF0 for(auto &v : g[u]) if(v != p)
B18 nxt[v] = (v == g[u][0] ? nxt[u] : v),
42C hld(v, u);
36C }

D41 /// OPTIONAL ///
310 int lca(int u, int v){
582 while(!inSubtree(nxt[u], v)) u = parent[nxt[u]];
E1D while(!inSubtree(nxt[v], u)) v = parent[nxt[v]];
40A return tin[u] < tin[v] ? u : v;
AEB }
65E bool inSubtree(int u, int v){ return tin[u] <= tin[v] &&
tin[v] < tin[u] + sz[u]; }
D41 //query/update_subtree[tin[u]+EDGE, tin[u]+sz[u]-1];
5D9 };
```

4.11 LCA

LCA - Lowest Common Ancestor - Binary Lifting
Algoritmo para encontrar o menor ancestral comum
entre dois vertices em uma arvore enraizada

IMPORTANTE! O algoritmo esta 0-indexado

Complexity:

buildBL() -> O(N Log N)

lca() -> O(Log N)

* Informacoes

-> chame dfs(root, root) para calcular o pai e a altura de
cada vertice
-> chame buildBL() para criar a matriz do Binary Lifting
-> chame lca(u, v) para encontrar o menor ancestral comum
bl[i][u] -> Binary Lifting com o (2^i)-esimo pai de u
lvl[u] -> Altura ou level de U na arvore

```

81D const int MAXN = 1e4 + 5;
633 const int MAXLG = 16;
```

```
A34 vector<int> grafo[MAXN];
```

```
A87 int bl[MAXLG][MAXN], lvl[MAXN];
```

```
80E void dfs(int u, int p, int l=0){
```

```

34C lvl[u] = 1;
4FB bl[0][u] = p;

F3E for(auto v : grafo[u])
F6B if(v != p)
0C5 dfs(v, u, l+1);
9A8 }

555 void buildBL(int N){
977 for(int i=1; i<MAXLG; i++)
51F for(int u=0; u<N; u++)
69C bl[i][u] = bl[i-1][bl[i-1][u]];
59A }

310 int lca(int u, int v){
DC4 if(lvl[u] < lvl[v]) swap(u, v);

D07 for(int i=MAXLG-1; i>=0; i--)
179 if(lvl[u] - (1<<i) >= lvl[v])
319 u = bl[i][u];

60E if(u == v) return u;

D07 for(int i=MAXLG-1; i>=0; i--)
BFA if(bl[i][u] != bl[i][v])
E01 u = bl[i][u],
4BC v = bl[i][v];

68E return bl[0][u];
381 }
```

4.12 MinCostMaxFlow

```

7C9 struct Aresta {
F0B int u, v; ll cap, cost;
FF2 Aresta(int u, int v, ll cap, ll cost) : u(u), v(v), cap(
cap), cost(cost) {}
1D9 };
```

```

6F3 struct MCMF {
878 const ll INF = numeric_limits<ll>::max();
6B0 int n, source, sink;
903 vector<vector<int>>> adj;
4DF vector<Aresta> edges;
39D vector<ll> dist, pot;
E3B vector<int> from;

D98 MCMF(int n, int source, int sink) : n(n), source(source)
, sink(sink) { adj.resize(n); pot.resize(n); }

3A2 void addAresta(int u, int v, ll cap, ll cost){
471 adj[u].push_back(edges.size());
986 edges.emplace_back(u, v, cap, cost);

282 adj[v].push_back(edges.size());
29F edges.emplace_back(v, u, 0, -cost);
D21 }
```

```

26A queue<int> q;
B57 vector<bool> vis;
791 bool SPFA(){
EF2 dist.assign(n, INF);
0B5 from.assign(n, -1);
543 vis.assign(n, false);
```

```
7CD q.push(source);
```

```

506 dist[source] = 0;

14D while(!q.empty()){
E4A int u = q.front();
833 q.pop();

776 vis[u] = false;

E20 for(auto i : adj[u]){
F42 if(edges[i].cap == 0) continue;
628 int v = edges[i].v;
99A ll cost = edges[i].cost;

148 if(dist[v] > dist[u] + cost + pot[u] - pot[v]){
DEC dist[v] = dist[u] + cost + pot[u] - pot[v];
203 from[v] = i;
A1A if(!vis[v]) q.push(v), vis[v] = true;
888 }
652 }
344 }

19E for(int u=0; u<n; u++) //fix pot
067 if(dist[u] < INF)
AB7 pot[u] += dist[u];

9DE return dist[sink] < INF;
D50 }

B84 pair<ll, ll> augment(){
B3F ll flow = edges[from[sink]].cap, cost = 0; //fixed
flow: flow = min(flow, remainder)

940 for(int v=sink; v != source; v = edges[from[v]].u)
73D flow = min(flow, edges[from[v]].cap),
871 cost += edges[from[v]].cost;

940 for(int v=sink; v != source; v = edges[from[v]].u)
86A edges[from[v]].cap -= flow,
674 edges[from[v]^1].cap += flow;

884 return {flow, cost};
668 }

164 bool inCut(int u){ return dist[u] < INF; }
```

```

6DC pair<ll, ll> maxFlow(){
D7D ll flow = 0, cost = 0;
```

```

4EB while( SPFA() ){
274 auto [f, c] = augment();
C87 flow += f;
BFC cost += f*c;
35C }
884 return {flow, cost};
D37 }
22E };
```

4.13 SCC - Kosaraju

Kosaraju - Strongly Connected Component
Algoritmo de Kosaraju para encontrar Componentes Fortemente
Conexas

Complexity: O(V + E)

IMPORTANTE! O algoritmo esta 0-indexado

* Variaveis e explicacoes *

```

int C    -> C e a quantidade de Componetes Conexas. As
          componetes estao numeradas de 0 a C-1
dag      -> Apos rodar o Kosaraju, o grafo das componentes
          conexas sera criado aqui
comp[u]  -> Diz a qual componente conexa U faz parte
order    -> Ordem de saida dos vertices. Necessario para o
          Kosaraju
grafo    -> grafo direcionado
greve    -> grafo reverso (que deve ser construido junto ao
          grafo normal) !!!

```

NOTA: A ordem que o Kosaraju descobre as componentes e uma Ordenacao Topologica do SCC em que o dag[0] nao possui grau de entrada e o dag[C-1] nao possui grau de saida

```

0C1 #define vi vector<int>

```

```

229 const int MAXN = 1e6 + 5;

```

```

C92 vi grafo[MAXN];
4ED vi greve[MAXN];
404 vi dag[MAXN];
104 vi comp, order;
B57 vector<bool> vis;
868 int C;

```

```

315 void dfs(int u){
B9C     vis[u] = true;
F3E     for(auto v : grafo[u])
C2D         if(!vis[v])
6B4             dfs(v);
C75     order.push_back(u);
8C4 }

```

```

163 void dfs2(int u){
361     comp[u] = C;
6A8     for(auto v : greve[u])
750         if(comp[v] == -1)
D5A             dfs2(v);
1F8 }

```

```

955 void kosaraju(int n){
070     order.clear();
E28     comp.assign(n, -1);
543     vis.assign(n, false);

```

```

84D     for(int v=0; v<n; v++){
C2D         if(!vis[v])
6B4             dfs(v);

```

```

796     C = 0;
3B9     reverse(begin(order), end(order));

```

```

961     for(auto v : order)
750         if(comp[v] == -1)
400             dfs2(v), C++;

```

```

D41     //// Montar DAG ////
78F     vector<bool> marc(C, false);

```

```

687     for(int u=0; u<n; u++){
F3E         for(auto v : grafo[u])
F95             {
264                 if(comp[v] == comp[u] || marc[comp[v]]) continue;

```

```

812                 marc[comp[v]] = true;
F26                 dag[comp[u]].emplace_back(comp[v]);
0DC             }

```

```

09D     for(auto v : grafo[u]) marc[comp[v]] = false;
A85     }
80A }

```

4.14 Tarjan

Tarjan - Pontes e Pontos de Articulacao
Algoritmo para encontrar pontes e pontos de articulacao.

Complexity: $O(V + E)$
IMPORTANTE! Lembre do `memset(pre, -1, sizeof pre);`

*** Variaveis e explicacoes ***
pre[u] = "Altura", ou, x-esimo elemento visitado na DFS.
Usado para saber a posicao de um vertice na arvore de DFS
low[u] = Low Link de U, ou a menor aresta de retorno (mais proxima da raiz) que U alcanca entre seus filhos

chd = Children. Quantidade de componentes filhos de U. Usado para saber se a Raiz e Ponto de Articulacao.
any = Marca se alguma aresta de retorno em qualquer dos componentes filhos de U nao ultrapassa U. Se isso for verdade, U e Ponto de Articulacao.

if(low[v] > pre[u]) pontes.emplace_back(u, v); -> se a mais alta aresta de retorno de V (ou o menor low) estiver abaixo de U, entao U-V e ponte
if(low[v] >= pre[u]) any = true; -> se a mais alta aresta de retorno de V (ou o menor low) estiver abaixo de U ou igual a U, entao U e Ponto de Articulacao

```

229 const int MAXN = 1e6 + 5;
F4C int pre[MAXN], low[MAXN], clk=0;
A34 vector<int> grafo [MAXN];

```

```

A2B vector<pair<int, int>> pontes;
252 vector<int> cut;

```

```

A76 #warning "lembrar do memset(pre, -1, sizeof pre);"
CF2 void tarjan(int u, int p = -1){
FD2     pre[u] = low[u] = clk++;

```

```

B10     bool any = false;
378     int chd = 0;

```

```

2BF     for(auto v : grafo[u]){
730         if(v == p) continue;

```

```

9BE         if(pre[v] == -1)
F95             {
3D2                 tarjan(v, u);

```

```

E7F         low[u] = min(low[v], low[u]);

```

```

334         if(low[v] > pre[u]) pontes.emplace_back(u, v);
23A         if(low[v] >= pre[u]) any = true;

```

```

87D         chd++;
302     }
295     else
201         low[u] = min(low[u], pre[v]);
6D3 }

```

```

B82 if(p == -1 && chd >= 2) cut.push_back(u);
5F3 if(p != -1 && any)     cut.push_back(u);
E3D }

```

5 Strings

5.1 hash2

String Hash - Double Hash
precalc() -> $O(N)$
StringHash() -> $O(|S|)$
gethash() -> $O(1)$

StringHash hash(s); -> Cria o Hash da string s
hash.gethash(l, r); -> Hash [L,R] (0-Indexado)

```

229 const int MAXN = 1e6 + 5;

```

```

E8E const ll MOD1 = 131'807'699;
D5D const ll MOD2 = 1e9 + 9;
145 const ll base = 157;

```

```

DB4 ll expb1[MAXN], expb2[MAXN];

```

```

921 #warning "Call precalc() before use StringHash"
FE8 void precalc(){
6D8     expb1[0] = expb2[0] = 1;

```

```

7E4     for(int i=1;i<MAXN;i++){
E0E         expb1[i] = expb1[i-1]*base % MOD1,
C4B         expb2[i] = expb2[i-1]*base % MOD2;
A02 }

```

```

3CE struct StringHash{
0DD     vector<pair<ll,ll>> hsh;
AC0     string s; // comment S if you dont need it

```

```

6F2     StringHash(string& s) : s(s){
63F         hsh.assign(s.size()+1, {0,0});

```

```

724         for (int i=0;i<s.size();i++){
B7A             hsh[i+1].first = ( hsh[i].first *base % MOD1
+ s[i] ) % MOD1,
08F             hsh[i+1].second = ( hsh[i].second*base % MOD2
+ s[i] ) % MOD2;
5A6         }

```

```

2F0     ll gethash(int a,int b){
F96         ll h1 = (MOD1+ hsh[b+1].first - hsh[a].first *
expb1[b-a+1] % MOD1) % MOD1;
F4A         ll h2 = (MOD2+ hsh[b+1].second - hsh[a].second*
expb2[b-a+1] % MOD2) % MOD2;
D23         return (h1<<32) | h2;
C77     }
1D3 };

```

```

FE3 int firstDiff(StringHash& a, int la, int ra, StringHash& b
, int lb, int rb)
F95 {
7E5     int l=0, r=min(ra-la, rb-lb), diff=r+1;
3D5     while(l <= r){
EE4         int m = (l+r)/2;
065         if(a.gethash(la, la+m) == b.gethash(lb, lb+m)) l = m
+1;
72D         else r = m-1, diff = m;
BAD     }
2B1     return diff;
9B7 }

```

```

03D int hshComp(StringHash& a, int la, int ra, StringHash& b,
    int lb, int rb){
E85     int diff = firstDiff(a, la, ra, b, lb, rb);
23E     if(diff > ra-la && ra-la == rb-lb) return 0; //equal
D15     if(diff > ra-la || diff > rb-lb) return ra-la < rb-lb
        ? -2 : +2; //prefix of the other
626     return a.s[la+diff] < b.s[lb+diff] ? -1 : +1;
8C4 }

```

5.2 KMP

```

F0A vector<int> pi(string &t){
82B     vector<int> p(t.size(), 0);

7FA     for(int i=1, j=0; i<t.size(); i++)
F95     {
90B         while(j > 0 && t[j] != t[i]) j = p[j-1];

3C7         if(t[j] == t[i]) j++;

F8C         p[i] = j;
C12     }

74E     return p;
238 }

2AD vector<int> kmp(string &s, string &t){
690     vector<int> p = pi(t), occ;

00A     for(int i=0, j=0; i<s.size(); i++)
F95     {
705         while( j > 0 && s[i] != t[j]) j = p[j-1];

566         if(s[i]==t[j]) j++;

2F0         if(j == t.size()) occ.push_back(i-j+1), j = p[j-1];
F09     }

FB0     return occ;
37B }

0F8 KMP - Knuth-Morris-Pratt Pattern Searching

05C Complexity: O(|S|+|T|)

7DD S -> String
8A6 T -> Pattern

```

5.3 Aho-Corasick

Aho-Corasick: Trie automaton to search multiple patterns in a text

Complexity: $O(\text{SUM}|P| + |S|) * \text{ALPHA}$

```

Aho<26,'a'> aho;
for(auto p: patterns) aho.add(p);
aho.buildSufixLink();
auto ans = aho.findPattern(s);

parent(p), sufuxLink(sl), outputLink(ol), patternID(idw)
outputLink -> edge to other pattern end (when p is a sufux of
it)

```

ALPHA -> Size of the alphabet. If big, consider changing nxt to map

To find ALL occurrences of all patterns, don't delete ol in findPattern. But it can be slow (at number of occ), so consider using DP on the automaton.

```

E8D template<const int ALPHA = 26, const int off = 'a'>
C99 struct Aho {
BF2     struct Node {
E05         Node* p = NULL;
A26         Node* sl = NULL;
C3A         Node* ol = NULL;
CB8         array<Node*, ALPHA> nxt;

7DE         char c;
BBC         int idw = -1;

212         Node() { nxt.fill(NULL); }
B04         Node(Node* p, char c) : p(p), c(c) { nxt.fill(NULL)
        }; }
92D     };
2CA     typedef Node* trie;

ACD     trie root;
EAA     int nwords = 0;
63B     Aho() { root = new Node(); }

22D     void add(string &s){
346         trie t = root;
242         for(auto c : s){ c -= off;
508             if(!t->nxt[c])
02F                 t->nxt[c] = new Node(t, c);
4F8             t = t->nxt[c];
E9A         }
71E         t->idw = nwords++; //cuidado com strings iguais!
        use vector
625     }

34A     void buildSufixLink(){
A2F         deque<trie> q(1, root);

14D         while(!q.empty()){
81D             trie t = q.front();
CED             q.pop_front();

630             if(trie w = t->p){
29D                 do w = w->sl; while(w && !w->nxt[t->c]);
619                 t->sl = w ? w->nxt[t->c] : root;
D7B                 t->ol = t->sl->idw == -1 ? t->sl->ol : t->
        sl;
8DB             }

806             for(int c=0; c<ALPHA; c++)
F72                 if(t->nxt[c])
78D                     q.push_back(t->nxt[c]);
693         }
09C     }

66F     vector<bool> findPattern(string &s){
BFD         vector<bool> ans(nwords, 0);
82D         trie w = root;
242         for(auto c : s){ c -= off;
A7A             while(w && !w->nxt[c]) w = w->sl;
AEA             w = w ? w->nxt[c] : root;

5BE             for(trie z=w, nl; z; nl=z->ol, z->ol=NULL, z=
        nl)
972                 if(z->idw != -1) //get ALL occ: dont
        delete ol (may slow)

```

```

31E             ans[z->idw] = true;
B04         }
BA7         return ans;
C8E     }
3D4 };

```

5.4 Manacher

```

DC6 vector<int> manacher(string &st){
E13     string s = "$_";
821     for(char c : st){ s += c; s += "_"; }
095     s += "#";

995     int n = s.size()-2;

BD7     vector<int> p(n+2, 0);
7CD     int l=1, r=1;

557     for(int i=1, j; i<=n; i++)
F95     {
DAF         p[i] = max(0, min(r-i, p[l+r-i])) ; //atualizo o valor
        atual para o valor do palindromo espelho na string ou
        para o total que esta contido

A5F         while( s[i-p[i]] == s[i+p[i]] ) p[i]++;

39C         if( i+p[i] > r ) l = i-p[i], r = i+p[i];
A83     }

6AE     for(auto &x : p) x--; //o valor de p[i] e igual ao
        tamanho do palindromo + 1

74E     return p;
907 }

BEF Manacher Algorithm
64E Find every palindrome in string
80E Complexidade: O(N)

```

5.5 trie

Trie - Arvore de Prefixos
 insert(P) - $O(|P|)$
 count(P) - $O(|P|)$
 MAXS - Soma do tamanho de todas as Strings
 sigma - Tamanho do alfabeto

```

AAF const int MAXS = 1e5 + 10;
70C const int sigma = 26;

F6C int trie[MAXS][sigma], terminal[MAXS], z = 1;

33B void insert(string &p){
B3D     int cur = 0;

E2E     for(int i=0; i<p.size(); i++){
1BF         int id = p[i] - 'a';

BCF         if(trie[cur][id] == -1 ){
616             memset(trie[z], -1, sizeof trie[z]);
869             trie[cur][id] = z++;
CAE         }

```

```

3AD     cur = trie[cur][id];
A9E   }

B07   terminal[cur]++;
C89   }

684 int count(string &p){
B3D   int cur = 0;

E2E   for(int i=0; i<p.size(); i++){
94B     int id = (p[i] - 'a');

C39     if(trie[cur][id] == -1) return 0;

3AD     cur = trie[cur][id];
ADB   }
89E   return terminal[cur];
D3C   }

CA2 void init(){
E6F   memset(trie[0], -1, sizeof trie[0]);
34E   z = 1;
A11   }

```

5.6 Z-Function

```

403 vector<int> Zfunction(string &s){ // O(N)
163   int n = s.size();
2B1   vector<int> z (n, 0);

A5C   for(int i=1, l=0, r=0; i<n; i++){
76D     if(i <= r) z[i] = min(z[i-l], r-i+1);

F61     while(z[i] + i < n && s[z[i]] == s[i+z[i]]) z[i]++;

EAF     if(r < i+z[i]-1) l = i, r = i+z[i]-1;
0CD   }

070   return z;
D58   }

```

6 others

6.1 Hungarian

Hungarian Algorithm - Assignment Problem
Algoritmo para o problema de atribuicao minima.

Complexity: $O(N^2 * M)$

hungarian(int n, int m); -> Retorna o valor do custo minimo
getAssignment(int m) -> Retorna a lista de pares
<linha, Coluna> do Minimum Assignment

n -> Numero de Linhas // m -> Numero de Colunas

IMPORTANTE! O algoritmo e 1-indexado
IMPORTANTE! O tipo padrao esta como int, para mudar para
outro tipo altere | typedef <TIPO> TP; |
Extra: Para o problema da atribuicao maxima, apenas
multiplique os elementos da matriz por -1

```

941 typedef int TP;

3CE const int MAXN = 1e3 + 5;
657 const TP INF = 0x3f3f3f3f;

F31 TP matrix[MAXN][MAXN];
F10 TP row[MAXN], col[MAXN];
E1F int match[MAXN], way[MAXN];

E5E TP hungarian(int n, int m){
715   memset(row, 0, sizeof row);
CD2   memset(col, 0, sizeof col);
187   memset(match, 0, sizeof match);

535   for(int i=1; i<=n; i++)
F95   {
96C     match[0] = i;
23B     int j0 = 0, j1, i0;
76E     TP delta;

693     vector<TP> minv (m+1, INF);
C04     vector<bool> used (m+1, false);

016     do {
472       used[j0] = true;
F81       i0 = match[j0];
B27       j1 = -1;
7DA       delta = INF;

2E2       for(int j=1; j<=m; j++)
F92       if(!used[j]){
76D         TP cur = matrix[i0][j] - row[i0] - col[j];

9F2         if( cur < minv[j] ) minv[j] = cur, way[j] = j0;
821         if(minv[j] < delta) delta = minv[j], j1 = j;
6FD       }

FC9       for(int j=0; j<=m; j++)
E48       if(used[j]){
7AC         row[match[j]] += delta,
429         col[j] -= delta;
72C       }else
299         minv[j] -= delta;

6D4       j0 = j1;
A95     } while(match[j0]);

016     do {
B8C       j1 = way[j0];
77A       match[j0] = match[j1];
6D4       j0 = j1;
196     } while(j0);
7B1   }

A33   return -col[0];
7FF   }

3B4 vector<pair<int, int>> getAssignment(int m){
F77   vector<pair<int, int>> ans;

8EA   for(int i=1; i<=m; i++)
843     ans.push_back(make_pair(match[i], i));

BA7   return ans;
01D   }

```

```

016   do {
B8C     j1 = way[j0];
77A     match[j0] = match[j1];
6D4     j0 = j1;
196   } while(j0);
7B1   }

A33   return -col[0];
7FF   }

3B4 vector<pair<int, int>> getAssignment(int m){
F77   vector<pair<int, int>> ans;

8EA   for(int i=1; i<=m; i++)
843     ans.push_back(make_pair(match[i], i));

BA7   return ans;
01D   }

```

6.2 MO

Algoritmo de MO para query em range

Complexity: $O((N + Q) * \text{SQRT}(N) * F)$ | F e a complexidade do Add e Remove

IMPORTANTE! Queries devem ter seus indices (Idx) 0-indexados!

Modifique as operacoes de Add, Remove e GetAnswer de acordo com o problema.
BLOCK_SZ pode ser alterado para aproximadamente $\text{SQRT}(\text{MAX_N})$

```

861 const int BLOCK_SZ = 700;

670 struct Query{
738   int l, r, idx;

991   Query(int l, int r, int idx) : l(l), r(r), idx(idx) {}

406   bool operator < (Query q) const {
6EB     if(l / BLOCK_SZ != q.l / BLOCK_SZ) return l < q.l;
387     return (l / BLOCK_SZ &1) ? ( r < q.r ) : ( r > q.r );
667   }
F51   };

543 void add(int idx);
F8A void remove(int idx);
AD7 int getAnswer();

73F vector<int> MO(vector<Query> &queries){
51F   vector<int> ans(queries.size());

BFA   sort(queries.begin(), queries.end());

32D   int L = 0, R = 0;
49E   add(0);

FE9   for(auto [l, r, idx] : queries){
128     while(l < L) add(--L);
C4A     while(r > R) add(++R);
684     while(l > L) remove(L++);
B50     while(r < R) remove(R--);

830     ans[idx] = getAnswer();
08D   }

BA7   return ans;
ACF   }

D41 /* IF you want to use hilbert curves on MO
0BD   vector<ll> h(ans.size());
CEC   for (int i = 0; i < ans.size(); i++) h[i] = hilbert(
queries[i].l, queries[i].r);
063   sort(queries.begin(), queries.end(), [&](Query&a, Query&b)
{ return h[a.idx] < h[b.idx]; }); */
E51   inline ll hilbert(int x, int y){
C85     static int N = 1 << (__builtin_clz(0) - __builtin_clz(
MAXN));
B69     int rx, ry, s; ll d = 0;
43B     for(s = N/2; s > 0; s /= 2){
C95       rx = (x & s) > 0, ry = (y & s) > 0;
F15       d += s * (ll)(s) * ((3 * rx) ^ ry);
E2D       if(ry == 0) { if(rx == 1) x = N-1 - x, y = N-1 - y;
swap(x, y); }
200     }
BE2     return d;
038   }

```

6.3 MOTree

Algoritmo de MO para query de caminho em arvore

Complexity: $O((N + Q) * \text{SQRT}(N) * F)$ | F e a complexidade do Add e Remove
IMPORTANTE! 0-indexado!

```
80E const int MAXN = 1e5+5;
F5A const int BLOCK_SZ = 500;
304 struct Query{int l, r, idx;}; //same of MO. Copy operator
<
```

```
282 vector<int> g[MAXN];
212 int tin[MAXN], tout[MAXN];
03B int pai[MAXN], order[MAXN];
```

```
179 void remove(int u);
C8B void add(int u);
AD7 int getAnswer();
```

```
C0A void go_to(int ti, int tp, int otp){
B21 int u = order[ti], v, to;
61E to = tout[u];
AA5 while(!(ti <= tp && tp <= to)){ //subo com U (ti) ate
ser ancestral de W
E7C v = pai[u];
```

```
BAF if(ti <= otp && otp <= to) add(v);
96E else remove(u);
```

```
A68 u = v;
363 ti = tin[u];
61E to = tout[u];
462 }
```

```
915 int w = order[tp];
D88 to = tout[w];
082 while(ti < tp){ //subo com W (tp) ate U
80E v = pai[w];
```

```
F19 if(tp <= otp && otp <= to) remove(v);
7AC else add(w);
```

```
9A1 w = v;
FCA tp = tin[w];
D88 to = tout[w];
34D }
B15 }
```

```
1D4 int TIME = 0;
FB6 void dfs(int u, int p){
49E pai[u] = p;
6FD tin[u] = TIME++;
A2B order[tin[u]] = u;
```

```
70D for(auto v : g[u])
F6B if(v != p)
95E dfs(v, u);
916 tout[u] = TIME-1;
686 }
```

```
73F vector<int> MO(vector<Query> &queries){
51F vector<int> ans(queries.size());
564 dfs(0, 0);
```

```
C89 for(auto &[u, v, i] : queries)
```

```
563 tie(u, v) = minmax(tin[u], tin[v]);
BFA sort(queries.begin(), queries.end());
```

```
49E add(0);
7AC int Lm = 0, Rm = 0;
FE9 for(auto [l, r, idx] : queries){
9D4 if(l < Lm) go_to(Lm, l, Rm), Lm = l;
0E8 if(r > Rm) go_to(Rm, r, Lm), Rm = r;
A5C if(l > Lm) go_to(Lm, l, Rm), Lm = l;
035 if(r < Rm) go_to(Rm, r, Lm), Rm = r;
830 ans[idx] = getAnswer();
30A }
```

```
BA7 return ans;
64A }
```

7 Math

7.1 fexp

```
11 MOD = 1e9 + 7;
```

```
11 fexp(ll b, ll p){
ll ans = 1;

while(p){
if(p&1) ans = (ans*b) % MOD;
b = b * b % MOD;
p >>= 1;

return ans % MOD;
}
// O(Log P) // b - Base // p - Potencia
```

7.2 CRT

```
D40 #define ld long double
```

```
2D3 ll modinverse(ll a, ll b, ll s0 = 1, ll s1 = 0) { return b
== 0 ? s0 : modinverse(b, a % b, s1, s0 - s1 * (a / b));
}
D8B ll mul(ll a, ll b, ll m) {
016 ll q = (long double) a * (long double) b / (long
double) m;
1A8 ll r = a * b - q * m;
B8B return (r + m) % m;
B9D }
```

```
28D struct Equation {
4C5 ll mod, ans;
08F bool valid;
0FC Equation() { valid = false; }
5E2 Equation(ll a, ll m) { mod = m, ans = (a % m + m) % m,
valid = true; }
4D3 Equation(Equation a, Equation b){
355 if(!a.valid || !b.valid){ valid = false; return; }
85C ll g = gcd(a.mod, b.mod);
DBE if((a.ans - b.ans) % g != 0){ valid = false;
return; }
```

```
AF0 valid = true;
```

```
B98 mod = a.mod * (b.mod / g);
2F6 ans = a.ans;
B8E ans += mul( mul(a.mod, modinverse(a.mod, b.mod),
mod), (b.ans - a.ans) / g, mod);

C4C ans = (ans % mod + mod) % mod;
F7C }
634 Equation operator+(const Equation& b) const { return
Equation(*this, b); }
D66 };
D41 // Equation eq1(2, 3); // x = 2 mod 3
D41 // Equation eq2(3, 5); // x = 3 mod 5
D41 // Equation ans = eq1 + eq2;
```

8 Theorems

8.1 Propriedades Matemáticas

- **Conjectura de Goldbach:** Todo número par $n > 2$ pode ser representado como $n = a + b$, onde a e b são primos.
- **Primos Gêmeos:** Existem infinitos pares de primos $p, p + 2$.
- **Conjectura de Legendre:** Sempre existe um primo entre n^2 e $(n + 1)^2$.
- **Lagrange:** Todo número inteiro pode ser representado como soma de 4 quadrados.
- **Zeckendorf:** Todo número pode ser representado como soma de números de Fibonacci diferentes e não consecutivos.
- **Tripla de Pitágoras (Euclides):** Toda tripla pitagórica primitiva pode ser gerada por $(n^2 - m^2, 2nm, n^2 + m^2)$ onde n e m são coprimos e um deles é par.
- **Wilson:** n é primo se e somente se $(n - 1)! \mod n = n - 1$.
- **Problema do McNugget:** Para dois coprimos x e y , o número de inteiros que não podem ser expressos como $ax + by$ é $(x - 1)(y - 1)/2$. O maior inteiro não representável é $xy - x - y$.
- **Fermat:** Se p é primo, então $a^{p-1} \equiv 1 \mod p$. Se x e m são coprimos e m primo, então $x^k \equiv x^{k \mod (m-1)} \mod m$. *Euler:* $x^{\varphi(m)} \equiv 1 \mod m$. $\varphi(m)$ é o totiente de Euler.
- **Teorema Chinês do Resto:** Dado um sistema de congruências:

$$x \equiv a_1 \mod m_1, \quad \dots, \quad x \equiv a_n \mod m_n$$

com m_i coprimos dois a dois. E seja $M_i = \frac{m_1 m_2 \dots m_n}{m_i}$ e $N_i = M_i^{-1} \mod m_i$. Então a solução é dada por:

$$x = \sum_{i=1}^n a_i M_i N_i$$

Outras soluções são obtidas somando $m_1 m_2 \dots m_n$.

- **Números de Catalan:** Exemplo: expressões de parênteses bem formadas. $C_0 = 1$, e:

$$C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i} = \frac{1}{n+1} \binom{2n}{n}$$

- **Bertrand (Ballot):** Com $p > q$ votos, a probabilidade de sempre haver mais votos do tipo A do que B até o fim é: $\frac{p-q}{p+q}$. Permitindo empates: $\frac{p+1-q}{p+1}$. Multiplicando pela combinação total $\binom{p+q}{q}$, obtém-se o número de possibilidades.
- **Linearidade da Esperança:** $E[aX+bY] = aE[X] + bE[Y]$
- **Variância:** $\text{Var}(X) = E[(X - \mu)^2] = E[X^2] - E[X]^2$
- **Progressão Geométrica:** $S_n = a_1 \cdot \frac{q^n - 1}{q - 1}$
- **Soma dos Cubos:** $\sum_{k=1}^n k^3 = \left(\sum_{k=1}^n k\right)^2$
- **Lindström-Gessel-Viennot:** A quantidade de caminhos disjuntos em um grid pode ser computada como o determinante da matriz do número de caminhos.
- **Lema de Burnside:** Número de colares diferentes (sem contar rotações), com m cores e comprimento n :

$$\frac{1}{n} \left(m^n + \sum_{i=1}^{n-1} m^{\gcd(i,n)} \right)$$

- **Inversão de Möbius:**

$$\sum_{d|n} \mu(d) = \begin{cases} 1, & n = 1 \\ 0, & \text{caso contrário} \end{cases}$$

- **Propriedades de Coeficientes Binomiais:**

$$\begin{aligned} \binom{N}{K} &= \binom{N}{N-K} = \frac{N}{K} \binom{N-1}{K-1} \\ \sum_{k=0}^m (-1)^k \binom{n}{k} &= (-1)^m \binom{n-1}{m} \\ \sum_{m=0}^n \binom{m}{k} &= \binom{n+1}{k+1} \\ \sum_{k=0}^m \binom{n+k}{k} &= \binom{n+m+1}{m} \\ \sum_{k=0}^n \binom{n}{k}^2 &= \binom{2n}{n} \\ \sum_{k=0}^n \binom{n}{k} &= 2^n \\ \sum_{k=0}^n k \binom{n}{k} &= n \cdot 2^{n-1} \\ \sum_{k=0}^n \binom{n-k}{k} &= F_{n+1} \end{aligned}$$

- **Identidades Clássicas:**

- **Hockey-stick:** $\sum_{i=r}^n \binom{i}{r} = \binom{n+1}{r+1}$
- **Vandermonde:** $\binom{m+n}{r} = \sum_{k=0}^r \binom{m}{k} \binom{n}{r-k}$

- **Distribuições de Probabilidade:**

- **Uniforme:** $X \in \{a, a+1, \dots, b\}$, $E[X] = \frac{a+b}{2}$
- **Binomial:** n tentativas com probabilidade p de sucesso:

$$P(X = x) = \binom{n}{x} p^x (1-p)^{n-x}, \quad E[X] = np$$

- **Geométrica:** Número de tentativas até o primeiro sucesso:

$$P(X = x) = (1-p)^{x-1} p, \quad E[X] = \frac{1}{p}$$

8.2 Geometria

- **Fórmula de Euler:** Em um grafo planar ou poliedro convexo, temos: $V - E + F = 2$ onde V é o número de vértices, E o número de arestas e F o número de faces.

- **Teorema de Pick:** Para polígonos com vértices em coordenadas inteiras:

$$\text{Área} = i + \frac{b}{2} - 1$$

onde i é o número de pontos interiores e b o número de pontos sobre o perímetro.

- **Teorema das Duas Orelhas (Two Ears Theorem):** Todo polígono simples com mais de três vértices possui pelo menos duas "orelhas"—vértices que podem ser removidos sem gerar interseções. A remoção repetida das orelhas resulta em uma triangulação do polígono.
- **Incentro de um Triângulo:** É o ponto de interseção das bissetrizes internas e centro da circunferência inscrita. Se a , b e c são os comprimentos dos lados opostos aos vértices $A(X_a, Y_a)$, $B(X_b, Y_b)$ e $C(X_c, Y_c)$, então o incentro (X, Y) é dado por:

$$X = \frac{aX_a + bX_b + cX_c}{a + b + c}, \quad Y = \frac{aY_a + bY_b + cY_c}{a + b + c}$$

- **Triangulação de Delaunay:** Uma triangulação de um conjunto de pontos no plano tal que nenhum ponto está dentro do círculo circunscrito de qualquer triângulo. Essa triangulação:

- Maximiza o menor ângulo entre todos os triângulos.
- Contém a árvore geradora mínima (MST) euclidiana como subconjunto.

- **Fórmula de Brahmagupta:** Para calcular a área de um quadrilátero cíclico (todos os vértices sobre uma circunferência), com lados a , b , c e d :

$$s = \frac{a + b + c + d}{2}, \quad \text{Área} = \sqrt{(s-a)(s-b)(s-c)(s-d)}$$

Se $d = 0$ (ou seja, um triângulo), ela se reduz à fórmula de Heron:

$$\text{Área} = \sqrt{(s-a)(s-b)(s-c)s}$$

8.3 Grafos

- **Fórmula de Euler (para grafos planares):**

$$V - E + F = 2$$

onde V é o número de vértices, E o número de arestas e F o número de faces.

- **Handshaking Lemma:** O número de vértices com grau ímpar em um grafo é par.

- **Teorema de Kirchhoff (contagem de árvores geradoras):** Monte a matriz M tal que:

$$M_{i,i} = \deg(i), \quad M_{i,j} = \begin{cases} -1 & \text{se existe aresta } i-j \\ 0 & \text{caso contrário} \end{cases}$$

O número de árvores geradoras (spanning trees) é o determinante de qualquer co-fator de M (remova uma linha e uma coluna).

- **Condições para Caminho Hamiltoniano:**

- **Teorema de Dirac:** Se todos os vértices têm grau $\geq n/2$, o grafo contém um caminho Hamiltoniano.
- **Teorema de Ore:** Se para todo par de vértices não adjacentes u e v , temos $\deg(u) + \deg(v) \geq n$, então o grafo possui caminho Hamiltoniano.

- **Algoritmo de Borůvka:** Enquanto o grafo não estiver conexo, para cada componente conexa escolha a aresta de menor custo que sai dela. Essa técnica constrói a árvore geradora mínima (MST).

- **Árvores:**

- Existem C_n árvores binárias com n vértices (C_n é o n -ésimo número de Catalan).
- Existem C_{n-1} árvores enraizadas com n vértices.
- **Fórmula de Cayley:** Existem n^{n-2} árvores com vértices rotulados de 1 a n .
- **Código de Prüfer:** Remova iterativamente a folha com menor rótulo e adicione o rótulo do vizinho ao código até restarem dois vértices.

- **Fluxo em Redes:**

- **Corte Mínimo:** Após execução do algoritmo de fluxo máximo, um vértice u está do lado da fonte se $\text{level}[u] \neq -1$.
- **Máximo de Caminhos Disjuntos:**
 - * **Arestas disjuntas:** Use fluxo máximo com capacidades iguais a 1 em todas as arestas.
 - * **Vértices disjuntos:** Divida cada vértice v em v_{in} e v_{out} , conectados por aresta de capacidade 1. As arestas que entram vão para v_{in} e as que saem saem de v_{out} .

- **Teorema de König:** Em um grafo bipartido:

Cobertura mínima de vértices = Matching máximo

O complemento da cobertura mínima de vértices é o conjunto independente máximo.

- **Coberturas:**

- * **Vertex Cover mínimo:** Os vértices da partição X que **não** estão do lado da fonte no corte mínimo, e os vértices da partição Y que **estão** do lado da fonte.
- * **Independent Set máximo:** Complementar da cobertura mínima de vértices.
- * **Edge Cover mínimo:** É N -matching, pegando as arestas do matching e mais quaisquer arestas restantes para cobrir os vértices descobertos.

- **Path Cover:**

- * **Node-disjoint path cover mínimo:** Duplicar vértices em tipo A e tipo B e criar grafo bipartido com arestas de $A \rightarrow B$. O path cover é N -matching.
- * **General path cover mínimo:** Criar arestas de $A \rightarrow B$ sempre que houver caminho de A para B no grafo. O resultado também é N -matching.

- **Teorema de Dilworth:** O path cover mínimo em um grafo dirigido acíclico é igual à **antichain máxima** (conjunto de vértices sem caminhos entre eles).

- **Teorema do Casamento de Hall:** Um grafo bipartido possui um matching completo do lado X se:

$$\forall W \subseteq X, \quad |W| \leq |\text{vizinhos}(W)|$$

- **Fluxo Viável com Capacidades Inferiores e Superiores:** Para rede sem fonte e sumidouro:

- * Substituir a capacidade de cada aresta por $C_{\text{upper}} - C_{\text{lower}}$
- * Criar nova fonte S e sumidouro T
- * Para cada vértice v , compute:

$$M[v] = \sum_{\text{arestas entrando}} C_{\text{lower}} - \sum_{\text{arestas saindo}} C_{\text{lower}}$$

- * Se $M[v] > 0$, adicione aresta (S, v) com capacidade $M[v]$; se $M[v] < 0$, adicione (v, T) com capacidade $-M[v]$.
- * Se todas as arestas de S estão saturadas no fluxo máximo, então um fluxo viável existe. O fluxo viável final é o fluxo computado mais os valores de C_{lower} .

8.4 DP

- **Divide and Conquer Optimization:** Utilizada em problemas do tipo:

$$dp[i][j] = \min_{k < j} \{dp[i-1][k] + C[k][j]\}$$

onde o objetivo é dividir o subsegmento até j em i segmentos com algum custo. A otimização é válida se:

$$A[i][j] \leq A[i][j+1]$$

onde $A[i][j]$ é o valor de k que minimiza a transição.

- **Knuth Optimization:** Aplicável quando:

$$dp[i][j] = \min_{i < k < j} \{dp[i][k] + dp[k][j]\} + C[i][j]$$

e a condição de monotonicidade é satisfeita:

$$A[i][j-1] \leq A[i][j] \leq A[i+1][j]$$

com $A[i][j]$ sendo o índice k que minimiza a transição.

- **Slope Trick:** Técnica usada para lidar com funções lineares por partes e convexas. A função é representada por pontos onde a derivada muda, que podem ser manipulados com multiset ou heap. Útil para manter o mínimo de funções acumuladas em forma de envelopes convexas.

- **Outras Técnicas e Truques Importantes:**

- **FFT (Fast Fourier Transform):** Convolução eficiente de vetores.
- **CHT (Convex Hull Trick):** Otimização para DP com funções lineares e monotonicidade.
- **Aliens Trick:** Técnica para binarizar o custo em problemas de otimização paramétrica (geralmente em problemas com limite no número de grupos/segmentos).
- **Bitset:** Utilizado para otimizações de espaço e tempo em DP de subconjuntos ou somas parciais, especialmente em problemas de mochila.

9 Extra

9.1 Hash Function

Call

```
g++ hash.cpp -o hash
hash < code.cpp
```

to get the hash of the code.

The hash ignores comments and whitespaces.

The hash of a line with `}` is the hash of all the code since the `{` that opens it. (is the hash of that context)

(Optional) To make letters upperCase: `for(auto&c:s)if('a'<=c) c^=32;`

```
DE3  string getHash(string s){
909      ofstream ip("temp.cpp"); ip << s; ip.close();
EE9      system("g++ -E -P -dD -fpreprocessed ./temp.cpp | tr -d
      '[:space:]' | md5sum > hsh.temp");
CEF      ifstream fo("hsh.temp"); fo >> s; fo.close();
A15      return s.substr(0, 3);
17A  }

604  int main_() {
973      string l, t;
3DA      vector<string> st(10);
C61      while(getline(cin, l)){
54F          t = l;
242          for(auto c : l)
F11              if(c == '{') st.push_back(""); else
2F0              if(c == '}') t = st.back() + l, st.pop_back();
C33          cout << getHash(t) + " " + l + "\n";
1ED          st.back() += t + "\n";
D1B      }
63A  }
```
