

Competitive Programming Notebook

Aguardando o PR adicionando HLD na QueryTree

Contents

1 DS	2	4 Primitives	20
1.1 Trie	2	4.1 Set Union Intersection	20
1.2 Treap Maletta	2	5 General	20
1.3 Trie Old	3	5.1 Next Permutation	20
1.4 Dsu	3	5.2 Xor Basis	20
1.5 Segtree	4	5.3 Min Priority Queue	21
1.6 Mergesorttree	4	5.4 Custom Unordered Map	21
1.7 Seghash	5	5.5 Flags	21
1.8 Seglazy	6	5.6 First True	21
1.9 Segtree Lazy Iterative	6	5.7 Kosaraju	21
1.10 Seglazystuctnode	7	5.8 Base Converter	21
1.11 Bigk	9	5.9 Overflow	22
1.12 Range Color Update	9	5.10 Random	22
1.13 Maxqueue	9	5.11 Get Subsets Sum Iterative	22
1.14 Triexor	10	5.12 Interactive	23
1.15 Bit2d	10	5.13 Xor 1 To N	23
1.16 Treap Cp	11	5.14 Input By File	23
1.17 Querytree	12	5.15 Template	23
1.18 Sparse	13	5.16 Mix Hash	23
1.19 Mex	13	5.17 Last True	23
1.20 Cht	13	6 Geometry	23
1.21 Ordered Set	14	6.1 Convex Hull	23
1.22 Manhattan Mst	14	7 Graph	24
1.23 Bit	15	7.1 Floyd Warshall	24
1.24 Psum2d	15	7.2 Ford Fulkerson	24
1.25 Kruskal	15	7.3 Dinic	25
2 String	16	7.4 Lca	26
2.1 Suffix Array	16	7.5 3sat	27
2.2 Hash	16	7.6 Dijkstra	27
2.3 Split	17	7.7 Has Negative Cycle	28
2.4 Trie Xor	17	7.8 2sat	28
2.5 Prefix Func	18	7.9 Min Cost Max Flow	29
2.6 Is Substring	18	7.10 Bfs	30
3 Math	18	8 DP	30
3.1 Fexp	18	8.1 Lcs	30
3.2 Sieve	18	8.2 Knapsack	31
3.3 Ifac	18	8.3 Lis Binary Search	31
3.4 Division Trick	18	8.4 Digit Dp 2	32
3.5 Factorization	18	8.5 Lis Segtree	32
3.6 Fft Quirino	19	8.6 Edit Distance	32
3.7 Divisors	19	8.7 Range Dp	32
3.8 Number Sum Product Of Divisors	19	8.8 Digit Dp	33
3.9 Is Prime	20		
3.10 Log Any Base	20		
3.11 Ceil	20		

1 DS

1.1 Trie

```

1 struct Trie {
2
3     struct Node {
4         map<char, Node> adj; // dÃ¡ pra trocar por
vector(26)
5         ll finishHere;
6
7         Node() {
8             finishHere = 0;
9         }
10
11         bool find(char c) {
12             return adj.find(c) != adj.end();
13         }
14     };
15 };
16
17 Node mainNode;
18
19 Trie(){
20     mainNode = Node();
21 }
22
23 void add(string &s) {
24     Node *curNode = &mainNode;
25
26     for(auto &c : s) {
27
28         if(!curNode->find(c)) {
29             curNode->adj[c] = Node();
30         }
31
32         curNode = &curNode->adj[c];
33     }
34
35     curNode->finishHere += 1;
36 }
37
38 void dfs(Node& node) {
39     for(auto &v : node.adj) {
40         dfs(v.ss);
41         // faz alguma coisa
42     }
43 }
44
45 void dfs() {
46     return dfs(mainNode);
47 }
48
49 bool search(string &s) {
50     Node* curNode = &mainNode;
51
52     for(auto &c : s) {
53         if(!curNode->find(c))
54             return false;
55
56         curNode = &curNode->adj[c];
57     }
58
59     return curNode->finishHere > 0;
60 }
61
62 void debugRec(Node node, int depth) {
63     for(auto &x : node.adj) {
64         cout << string(3 * depth, ' ') << x.ff <<
65         " " << x.ss.finishHere << "\n";
66         debugRec(x.ss, depth + 1);
67     }
68 }

```

```

67     }
68
69     void debug() {
70         debugRec(mainNode, 0);
71     }
72
73 };

```

1.2 Treap Maletta

```

1 // CÃdigo do Bruno Maletta!!!!!!
2 // pra problemas mais simples, usar a treap do cp!
3 // essa aqui Ã mais poderosa, mas por isso Ã um
pouco mais lenta
4
5
6 // Treap Implicita
7 //
8 // Todas as operacoes custam
9 // O(log(n)) com alta probabilidade
10
11 mt19937 rng((int) chrono::steady_clock::now().
time_since_epoch().count());
12
13 template<typename T> struct treap {
14     struct node {
15         node *l, *r;
16         int p, sz;
17         T val, sub, lazy;
18         bool rev;
19         node(T v) : l(NULL), r(NULL), p(rng()), sz(1)
, val(v), sub(v), lazy(0), rev(0) {}
20         void prop() {
21             if (lazy) {
22                 val += lazy, sub += lazy*sz;
23                 if (l) l->lazy += lazy;
24                 if (r) r->lazy += lazy;
25             }
26             if (rev) {
27                 swap(l, r);
28                 if (l) l->rev ^= 1;
29                 if (r) r->rev ^= 1;
30             }
31             lazy = 0, rev = 0;
32         }
33         void update() {
34             sz = 1, sub = val;
35             if (l) l->prop(), sz += l->sz, sub += l->
sub;
36             if (r) r->prop(), sz += r->sz, sub += r->
sub;
37         }
38     };
39
40     node* root;
41
42     treap() { root = NULL; }
43     treap(const treap& t) {
44         throw logic_error("Nao copiar a treap!");
45     }
46     ~treap() {
47         vector<node*> q = {root};
48         while (q.size()) {
49             node* x = q.back(); q.pop_back();
50             if (!x) continue;
51             q.push_back(x->l), q.push_back(x->r);
52             delete x;
53         }
54     }
55
56     int size(node* x) { return x ? x->sz : 0; }
57     int size() { return size(root); }

```

1.3 Trie Old

```

58 void join(node* l, node* r, node*& i) { // assume
    que l < r
59     if (!l or !r) return void(i = l ? l : r);
60     l->prop(), r->prop();
61     if (l->p > r->p) join(l->r, r, l->r), i = l;
62     else join(l, r->l, r->l), i = r;
63     i->update();
64 }
65 void split(node* i, node*& l, node*& r, int v,
int key = 0) {
66     if (!i) return void(r = l = NULL);
67     i->prop();
68     if (key + size(i->l) < v) split(i->r, i->r, r
, v, key+size(i->l)+1), l = i;
69     else split(i->l, l, i->l, v, key), r = i;
70     i->update();
71 }
72 void push_back(T v) {
73     node* i = new node(v);
74     join(root, i, root);
75 }
76 T query(int l, int r) {
77     node *L, *M, *R;
78     split(root, M, R, r+1), split(M, L, M, l);
79     T ans = M->sub;
80     join(L, M, M), join(M, R, root);
81     return ans;
82 }
83 void update(int l, int r, T s) {
84     node *L, *M, *R;
85     split(root, M, R, r+1), split(M, L, M, l);
86     M->lazy += s;
87     join(L, M, M), join(M, R, root);
88 }
89 void reverse(int l, int r) {
90     node *L, *M, *R;
91     split(root, M, R, r+1), split(M, L, M, l);
92     M->rev ^= 1;
93     join(L, M, M), join(M, R, root);
94 }
95 };
96
97 // https://cses.fi/problemset/task/2074/
98 // Nesse problema vc tem que printar a soma de l...r
99 // e tmb dar um reverse no range l...r
100 void solve() {
101
102     int n, q;
103     cin >> n >> q;
104
105     treap<ll> root;
106
107     for(int i = 0; i < n; i++) {
108         ll re; cin >> re;
109         // coloca esse vertice no final do array (que
110         tã armazenado na treap)
111         root.push_back(re);
112     }
113
114     while(q--) {
115         int t, l, r;
116         cin >> t >> l >> r;
117         l--; r--;
118         if(t == 1) {
119             root.reverse(l, r);
120         } else {
121             cout << root.query(l, r) << "\n";
122         }
123     }
124 }

```

```

1 struct Trie {
2
3     int nxt = 1, sz, maxLet = 26; //tamanho do
    alfabeto
4     vector< vector<int> > trie;
5     bitset<(int)1e7> finish; //modificar esse valor
    pra ser >= n
6     //garantir que vai submeter em cpp 64
7
8     Trie(int n){
9         sz = n;
10        trie.assign(sz, vector<int>(maxLet,0));
11    }
12
13    void add(string &s){
14        int cur = 0;
15        for(auto c: s){
16            //alterar esse azinho dependendo da
    entrada!!
17            if(trie[cur][c-'a'] == 0){
18                trie[cur][c-'a'] = nxt++;
19                cur = trie[cur][c-'a'];
20            } else {
21                cur = trie[cur][c-'a'];
22            }
23        }
24        finish[cur] = 1;
25    }
26
27    int search(string& s){
28        int cur = 0;
29        for(auto c: s){
30            if(trie[cur][c - 'a'] == 0){
31                return 0;
32            }
33            cur = trie[cur][c-'a'];
34        }
35        return finish[cur];
36    }
37
38 };

```

1.4 Dsu

```

1 // DSU
2 //
3 // https://judge.yosupo.jp/submission/126864
4
5 struct DSU {
6     int n = 0, components = 0;
7     vector<int> parent;
8     vector<int> size;
9
10    DSU(int nn){
11        n = nn;
12        components = n;
13        size.assign(n + 5, 1);
14        parent.assign(n + 5, 0);
15        iota(parent.begin(), parent.end(), 0);
16    }
17
18    int find(int x){
19        if(x == parent[x]) {
20            return x;
21        }
22        //path compression
23        return parent[x] = find(parent[x]);
24    }
25
26    void join(int a, int b){

```

```

27     a = find(a);
28     b = find(b);
29
30     if(a == b) {
31         return;
32     }
33
34     if(size[a] < size[b]) {
35         swap(a, b);
36     }
37
38     parent[b] = a;
39     size[a] += size[b];
40     components -= 1;
41 }
42
43 int sameSet(int a, int b) {
44     a = find(a);
45     b = find(b);
46     return a == b;
47 }
48 };

```

1.5 Segtree

```

1 struct Segtree {
2
3     int n; //size do array que a seg vai ser criada
4     em cima
5     vector<ll> seg;
6
7     Segtree(vector<ll>& s){
8         n = (int)s.size();
9         seg.resize(n+n+n+n, 0);
10        seg_build(1,0,n-1,s);
11    }
12
13    ll merge(ll a, ll b){
14        //return a+b;
15        if(!a) a = 00;
16        if(!b) b = 00;
17        return min(a,b);
18    }
19
20    void seg_build(int x, int l, int r, vector<ll>& s)
21    {
22        if(r < l) return;
23        if(l == r){
24            seg[x] = s[l];
25        } else {
26            int mid = 1 + (r-l)/2;
27            seg_build(x+x, l, mid, s);
28            seg_build(x+x+1, mid+1, r, s);
29            seg[x] = merge(seg[x+x], seg[x+x+1]);
30        }
31
32        //nÃs atual, intervalo na Ãrvore e intervalo
33        pedido
34        ll q(int x, int l, int r, int i, int j){
35            if(r < i || l > j) return 0;
36            if(l >= i && r <= j) return seg[x];
37            int mid = 1 + (r-l)/2;
38            return merge(q(x+x,l,mid,i,j), q(x+x+1,mid+1,
39            r,i,j));
40        }
41
42        //att posi pra val
43        void att(int x, int l, int r, int posi, ll val){
44            if(l == r){
45                seg[x] = val;
46            } else {
47                int mid = 1 + (r-l)/2;

```

```

45        if(posi <= mid)att(x+x,l,mid,posi,val);
46        else att(x+x+1,mid+1,r,posi,val);
47        seg[x] = merge(seg[x+x], seg[x+x+1]);
48    }
49 }
50
51 int findkth(int x, int l, int r, int k){
52     if(l == r){
53         return l;
54     } else {
55         int mid = 1 + (r-l)/2;
56         if(seg[x+x] >= k){
57             return findkth(x+x,l,mid,k);
58         } else {
59             return findkth(x+x+1,mid+1, r, k -
60             seg[x+x]);
61         }
62     }
63 }
64
65 ll query(int l, int r){
66     return q(1, 0, n-1, l, r);
67 }
68
69 void update(int posi, ll val){ //alterar em posi
70     pra val
71     att(1, 0, n-1, posi, val);
72 }
73
74 int findkth(int k){
75     //kth smallest, 0(logN)
76     //use position i to count how many times
77     value 'i' appear
78     //merge must be the sum of nodes
79     return findkth(1,0,n-1,k);
80 }
81
82 };

```

1.6 Mergesorttree

```

1 //const int MAXN = 3e5 + 10;
2 //vector<int> seg[ 4 * MAXN + 10];
3
4 struct MergeSortTree {
5
6     int n; //size do array que a seg vai ser criada
7     em cima
8     vector< vector<int> > seg;
9     //vector< vector<ll> > ps; //prefix sum
10
11    MergeSortTree(vector<int>& s){
12        //se o input for grande (ou o tempo mt puxado
13        ), coloca a seg com size
14        //maximo de forma global
15        n = (int)s.size();
16        seg.resize(4 * n + 10);
17        //ps.resize(4 * n + 10);
18        seg_build(1,0,n-1,s);
19    }
20
21    vector<int> merge(vi& a, vi& b){
22        int i = 0, j = 0, p = 0;
23        vi ans(a.size() + b.size());
24        while(i < (int)a.size() && j < (int)b.size())
25        {
26            if(a[i] < b[j]){
27                ans[p++] = a[i++];
28            } else {
29                ans[p++] = b[j++];
30            }
31        }
32        while(i < (int)a.size()){

```

```

30         ans[p++] = a[i++];
31     }
32     while(j < (int)b.size()){
33         ans[p++] = b[j++];
34     }
35     return ans;
36 }
37
38 vector<ll> calc(vi& s) {
39     ll sum = 0;
40     vector<ll> tmp;
41     for(auto &x : s) {
42         sum += x;
43         tmp.push_back(sum);
44     }
45     return tmp;
46 }
47
48 void seg_build(int x, int l, int r, vector<int>&
s){
49     if(r < l) return;
50     if(l == r){
51         seg[x].push_back(s[l]);
52         //ps[x] = {s[l]};
53     } else {
54         int mid = l + (r-l)/2;
55         seg_build(x+x, l, mid, s);
56         seg_build(x+x+1, mid+1, r, s);
57         seg[x] = merge(seg[x+x], seg[x+x+1]);
58         //ps[x] = calc(seg[x]);
59     }
60 }
61
62 //nÃs atual, intervalo na Ãrvore e intervalo
pedido
63 // retorna a quantidade de numeros <= val em [l,
r]
64
65 ll q(int x, int l, int r, int i, int j, int val){
66     if(r < i || l > j) return 0;
67     if(l >= i && r <= j){
68         return (lower_bound(seg[x].begin(), seg[x
].end(), val) - seg[x].begin());
69     }
70     int mid = l + (r-l)/2;
71     return q(x+x, l, mid, i, j, val) + q(x+x+1, mid+1,
r, i, j, val);
72 }
73
74 // retorna a soma dos numeros <= val em [l, r]
75 // nÃs atual, intervalo na Ãrvore e intervalo
pedido
76 /*
77 ll q(int x, int l, int r, int i, int j, ll val){
78     if(r < i || l > j) return 0;
79     if(l >= i && r <= j){
80         auto it = upper_bound(seg[x].begin(), seg
[x].end(), val) - seg[x].begin();
81
82         if(val > seg[x].back()) {
83             return ps[x].back();
84         }
85
86         if(val < seg[x][0]) {
87             return 0;
88         }
89
90         return ps[x][it - 1];
91     }
92 }
93
94 int mid = l + (r-l)/2;
95

```

```

96         return q(x+x, l, mid, i, j, val) + q(x+x+1, mid+1,
r, i, j, val);
97     }
98     */
99
100 ll query(int l, int r, ll val){
101     return q(1, 0, n-1, l, r, val);
102 }
103
104 };

```

1.7 Seghash

```

1 template<typename T> //use as SegtreeHash<int> h or
SegtreeHash<char>
2 struct SegtreeHash {
3
4     int n; //size do array que a seg vai ser criada
em cima
5
6     // P = 31, 53, 59, 73 .... (prime > number of
different characters)
7     // M = 578398229, 895201859, 1e9 + 7, 1e9 + 9 (
big prime)
8     int p, m;
9
10    vector<ll> seg, pot;
11
12    ll minValue = 0; // menor valor possÃvel que
pode estar na estrutura
13    // isso Ã pra evitar que a hash
de '0' seja igual a de '0000...'
14
15    SegtreeHash(vector<T>& s, ll P = 31, ll MOD = (11
)1e9 + 7){
16        n = (int)s.size();
17        p = P; m = MOD;
18        seg.resize(4 * n, -1);
19        pot.resize(4 * n);
20        pot[0] = 1;
21        for(int i = 1; i < (int)pot.size(); i++) {
22            pot[i] = (pot[i - 1] * P) % MOD;
23        }
24        seg_build(1, 0, n - 1, s);
25    }
26
27    ll merge(ll a, ll b, int tam){
28        if(a == -1) return b;
29        if(b == -1) return a;
30        return (a + b * pot[tam]) % m;
31    }
32
33    void seg_build(int x, int l, int r, vector<T>& s)
{
34        if(r < l) return;
35        if(l == r){
36            seg[x] = (int)s[l] - minValue + 1;
37        } else {
38            int mid = l + (r-l)/2;
39            seg_build(x+x, l, mid, s);
40            seg_build(x+x+1, mid+1, r, s);
41            seg[x] = merge(seg[x+x], seg[x+x+1], mid
- l + 1);
42        }
43    }
44
45    //nÃs atual, intervalo na Ãrvore e intervalo
pedido
46    ll q(int x, int l, int r, int i, int j){
47        if(r < i || l > j) return -1;
48        if(l >= i && r <= j) return seg[x];
49        int mid = l + (r-l)/2;

```

```

50     return merge(q(x+x,l,mid,i,j), q(x+x+1,mid+1,
51 r,i,j), mid - max(i, l) + 1);
52 }
53 //att posi pra val
54 void att(int x, int l, int r, int posi, T val){
55     if(l == r){
56         seg[x] = (int)val - minValue + 1;
57     } else {
58         int mid = l + (r-1)/2;
59         if(posi <= mid)att(x+x,l,mid,posi,val);
60         else att(x+x+1,mid+1,r,posi,val);
61         seg[x] = merge(seg[x+x], seg[x+x+1], mid
- l + 1);
62     }
63 }
64
65 ll query(int l, int r){
66     return q(1, 0, n-1, l, r);
67 }
68
69 void update(int posi, T val){ //alterar em posi
pra val
70     att(1, 0, n-1, posi, val);
71 }
72
73 };

```

1.8 Seglazy

```

1 struct SegLazy {
2
3     int n;
4     vector<ll> seg;
5     vector<ll> lazy;
6
7     SegLazy(vector<ll>& arr){
8         n = (int)arr.size();
9         seg.assign(n+n+n+n, 0);
10        lazy.assign(n+n+n+n, 0);
11        build(1,0,n-1,arr);
12    }
13
14    ll merge(ll a, ll b){
15        return a+b;
16    }
17
18    void build(ll x, int l, int r, vector<ll>& arr){
19        if(l == r){
20            seg[x] = 1LL * arr[l];
21        } else {
22            int mid = l + (r-1)/2;
23            build(x+x, l, mid, arr);
24            build(x+x+1, mid+1, r, arr);
25            seg[x] = merge(seg[x+x], seg[x+x+1]);
26        }
27    }
28
29    void upd_lazy(ll node, ll l, ll r){
30        seg[node] += (ll)(r-l+1) * lazy[node];
31        ll esq = node + node, dir = esq + 1;
32
33        if(dir < (int)seg.size()){
34            lazy[esq] += lazy[node];
35            lazy[dir] += lazy[node];
36        }
37
38        lazy[node] = 0;
39    }
40
41    ll q(ll x, int l, int r, int i, int j){
42        upd_lazy(x,l,r);
43    }

```

```

44        if(r < i || l > j)
45            return 0;
46
47        if(l >= i && r <= j )
48            return seg[x];
49
50        int mid = l + (r-1)/2;
51        return merge(q(x+x,l,mid,i,j), q(x+x+1,mid+1,
r,i,j));
52    }
53
54    ll query(int l, int r){ //valor em uma posi
específica -> query de [l,l];
55        return q(1, 0, n-1, l, r);
56    }
57
58    void upd(ll x, int l, int r, int i, int j, ll u){
59        upd_lazy(x,l,r);
60        if(r < i || l > j) return;
61        if(l >= i && r <= j){
62            lazy[x] += u;
63            upd_lazy(x,l,r);
64        } else {
65            int mid = l + (r-1)/2;
66            upd(x+x,l,mid,i,j,u);
67            upd(x+x+1,mid+1,r,i,j,u);
68            seg[x] = merge(seg[x+x], seg[x+x+1]);
69        }
70    }
71
72    void upd_range(int l, int r, ll u){ //intervalo e
valor
73        upd(1,0,n-1,l,r,u);
74    }
75
76 };

```

1.9 Segtree Lazy Iterative

```

1 // Segtree iterativa com lazy
2 //
3 // https://codeforces.com/gym/103708/problem/C
4 //
5 // O(N * log(N)) build
6 // O(log(N)) update e query
7
8 const int MAX = 524288; // NEED TO BE POWER OF 2 !!!
9 const int LOG = 19; // LOG = ceil(log2(MAX))
10
11 namespace seg {
12     ll seg[2*MAX], lazy[2*MAX];
13     int n;
14
15     ll junta(ll a, ll b) {
16         return a+b;
17     }
18
19     // soma x na posicao p de tamanho tam
20     void poe(int p, ll x, int tam, bool prop=1) {
21         seg[p] += x*tam;
22         if (prop and p < n) lazy[p] += x;
23     }
24
25     // atualiza todos os pais da folha p
26     void sobe(int p) {
27         for (int tam = 2; p /= 2; tam *= 2) {
28             seg[p] = junta(seg[2*p], seg[2*p+1]);
29             poe(p, lazy[p], tam, 0);
30         }
31     }
32
33     void upd_lazy(int i, int tam) {
34         if (lazy[i] && (2 * i + 1) < 2 * MAX) {

```

```

35         poe(2*i, lazy[i], tam);
36         poe(2*i+1, lazy[i], tam);
37         lazy[i] = 0;
38     }
39 }
40
41 // propaga o caminho da raiz ate a folha p
42 void prop(int p) {
43     int tam = 1 << (LOG-1);
44     for (int s = LOG; s; s--, tam /= 2) {
45         int i = p >> s;
46         upd_lazy(i, tam);
47     }
48 }
49
50 void build(int n2) {
51     n = n2;
52     for (int i = 0; i < n; i++) seg[n+i] = 0;
53     for (int i = n-1; i; i--) seg[i] = junta(seg
54 [2*i], seg[2*i+1]);
55     for (int i = 0; i < 2*n; i++) lazy[i] = 0;
56 }
57
58 ll query(int a, int b) {
59     ll ret = 0;
60     for (prop(a+=n), prop(b+=n); a <= b; ++a/=2,
61 --b/=2) {
62         if (a%2 == 1) ret = junta(ret, seg[a]);
63         if (b%2 == 0) ret = junta(ret, seg[b]);
64     }
65     return ret;
66 }
67
68 void update(int a, int b, int x) {
69     int a2 = a += n, b2 = b += n, tam = 1;
70     for (; a <= b; ++a/=2, --b/=2, tam *= 2) {
71         if (a%2 == 1) poe(a, x, tam);
72         if (b%2 == 0) poe(b, x, tam);
73     }
74     sobe(a2), sobe(b2);
75 }
76
77 int findkth(int x, int l, int r, ll k, int tam){
78     int esq = x + x;
79     int dir = x + x + 1;
80
81     upd_lazy(x, tam);
82     upd_lazy(esq, tam/2);
83     upd_lazy(dir, tam/2);
84
85     if(l == r){
86         return l;
87     } else {
88         int mid = l + (r-l)/2;
89
90         if(seg[esq] >= k){
91             return findkth(esq,l,mid,k, tam/2);
92         } else {
93             return findkth(dir,mid+1, r, k - seg[
94 esq], tam/2);
95         }
96     }
97 }
98
99 int findkth(ll k){
100     // kth smallest, 0(logN)
101     // use position i to count how many times
102     value 'i' appear
103     // merge must be the sum of nodes
104     return findkth(1,0,n-1,k,(1 << (LOG-1)));
105 }
106 };

```

1.10 Seglazystuctnode

```

1 struct Node {
2
3     int l, r;
4
5     int pref0, suf0, best0;
6     int pref1, suf1, best1;
7
8     Node(){
9         pref0 = 0; suf0 = 0; best0 = 0;
10        pref1 = 0; suf1 = 0; best1 = 0;
11        l = -1; r = -1;
12    };
13
14    void Init(int val_, int l_, int r_) {
15        best0 = !val_;
16        pref0 = !val_;
17        suf0 = !val_;
18
19        best1 = val_;
20        pref1 = val_;
21        suf1 = val_;
22
23        l = l_;
24        r = r_;
25    }
26
27    bool AllZero() {
28        return r - l + 1 == best0;
29    }
30
31    bool AllOne() {
32        return r - l + 1 == best1;
33    }
34
35    void Reverse() {
36        swap(pref0, pref1);
37        swap(suf0, suf1);
38        swap(best0, best1);
39    }
40
41 };
42
43 Node Merge(Node a, Node b) {
44
45     if(a.l == -1 || a.r == -1) {
46         return b;
47     }
48
49     if(b.l == -1 || b.r == -1) {
50         return a;
51     }
52
53     auto ans = Node();
54
55     ans.l = a.l;
56     ans.r = b.r;
57
58     // -----
59     //
60
61     if(a.AllZero()) {
62         ans.pref0 = a.pref0 + b.pref0;
63     } else {
64         ans.pref0 = a.pref0;
65     }
66
67     if(b.AllZero()) {
68         ans.suf0 = b.suf0 + a.suf0;
69     } else {
70

```

```

71     ans.suf0 = b.suf0;
72 }
73
74 ans.best0 = max({
75     a.best0,
76     b.best0,
77     a.suf0 + b.pref0
78 });
79
80 // -----
81 //
82
83 if(a.AllOne()) {
84     ans.pref1 = a.pref1 + b.pref1;
85 } else {
86     ans.pref1 = a.pref1;
87 }
88
89 if(b.AllOne()) {
90     ans.suf1 = b.suf1 + a.suf1;
91 } else {
92     ans.suf1 = b.suf1;
93 }
94
95 ans.best1 = max({
96     a.best1,
97     b.best1,
98     a.suf1 + b.pref1
99 });
100
101 // -----
102 //
103
104 return ans;
105 }
106
107 struct SegLazy {
108
109     private:
110
111     int n;
112     vector<Node> seg;
113     vector<bool> lazy; // precisa reverter ou nao
114
115     void build(ll x, int l, int r, string& s){
116         if(l == r){
117             int val = s[l] - '0';
118             seg[x].Init(val, l, r);
119         } else {
120             int mid = l + (r-l)/2;
121             build(x+x, l, mid, s);
122             build(x+x+1, mid+1, r, s);
123             seg[x] = Merge(seg[x+x], seg[x+x+1]);
124         }
125     }
126
127     void upd_lazy(ll node, ll l, ll r){
128
129         if(lazy[node]) {
130             seg[node].Reverse();
131         }
132
133         ll esq = node + node, dir = esq + 1;
134
135         if(dir < (int)seg.size() && lazy[dir]){
136             lazy[esq] = !lazy[esq];
137             lazy[dir] = !lazy[dir];
138         }
139
140         lazy[node] = 0;
141
142     }
143
144     Node q(ll x, int l, int r, int i, int j){
145         upd_lazy(x,l,r);
146
147         if(r < i || l > j)
148             return Node();
149
150         if(l >= i && r <= j )
151             return seg[x];
152
153         int mid = l + (r-l)/2;
154         return Merge(q(x+x,l,mid,i,j), q(x+x+1,
155             mid+1,r,i,j));
156     }
157
158     void upd(ll x, int l, int r, int i, int j){
159         upd_lazy(x,l,r);
160         if(r < i || l > j) return;
161         if(l >= i && r <= j){
162             lazy[x] = !lazy[x];
163             upd_lazy(x,l,r);
164         } else {
165             int mid = l + (r-l)/2;
166             upd(x+x,l,mid,i,j);
167             upd(x+x+1,mid+1,r,i,j);
168             seg[x] = Merge(seg[x+x], seg[x+x+1]);
169         }
170     }
171
172     public:
173
174     SegLazy(string& s){
175         n = (int)s.size();
176         seg.assign(n+n+n+n, Node());
177         lazy.assign(n+n+n+n, 0);
178         build(1,0,n-1,s);
179     }
180
181     void update(int l){
182         upd(1,0,n-1,l,l);
183     }
184
185     void update_range(int l, int r){
186         upd(1,0,n-1,l,r);
187     }
188
189     Node query(int l){
190         return q(1, 0, n-1, l, l);
191     }
192
193     Node query(int l, int r){
194         return q(1, 0, n-1, l, r);
195     }
196
197 };
198
199 void solve() {
200     int n, q;
201     string s;
202
203     cin >> n >> q >> s;
204
205     SegLazy seg(s);
206
207     while(q--) {
208         int c, l, r;
209         cin >> c >> l >> r;
210
211         if(c == 1) {

```



```

214         // inverte l...r
215         seg.update_range(l - 1, r - 1);
216     } else {
217         // query l...r
218         auto node = seg.query(l - 1, r - 1);
219         cout << node.best1 << "\n";
220     }
221 }
222 }
223 }
224 }

```

1.11 Bigk

```

1 struct SetSum {
2     ll sum;
3     multiset<ll> ms;
4
5     SetSum() {}
6
7     void add(ll x) {
8         sum += x;
9         ms.insert(x);
10    }
11
12    int rem(ll x) {
13        auto it = ms.find(x);
14
15        if (it == ms.end()) {
16            return 0;
17        }
18
19        sum -= x;
20        ms.erase(it);
21        return 1;
22    }
23
24    ll getMin() { return *ms.begin(); }
25
26    ll getMax() { return *ms.rbegin(); }
27
28    ll getSum() { return sum; }
29
30    int size() { return (int)ms.size(); }
31 };
32
33 struct BigK {
34     int k;
35     SetSum gt, mt;
36
37     BigK(int k): k(k) {}
38
39     void balance() {
40         while (gt.size() > k) {
41             ll mn = gt.getMin();
42             gt.rem(mn);
43             mt.add(mn);
44         }
45
46         while (gt.size() < k && mt.size() > 0) {
47             ll mx = mt.getMax();
48             mt.rem(mx);
49             gt.add(mx);
50         }
51     }
52
53     void add(ll x) {
54         gt.add(x);
55         balance();
56     }
57
58     void rem(ll x) {
59         if (mt.rem(x) == 0) {

```

```

60             gt.rem(x);
61         }
62
63         balance();
64     }
65
66     // be careful, 0(abs(oldK - newK) * log)
67     void setK(int _k) {
68         k = _k;
69         balance();
70     }
71
72     // O(log)
73     void incK() { setK(k + 1); }
74
75     // O(log)
76     void decK() { setK(k - 1); }
77 };

```

1.12 Range Color Update

```

1 // Range color update (brunomaletta)
2 //
3 // update(l, r, c) colore o range [l, r] com a cor c,
4 // e retorna os ranges que foram coloridos {l, r, cor}
5 // query(i) retorna a cor da posicao i
6 //
7 // Complexidades (para q operacoes):
8 // update - O(log(q)) amortizado
9 // query - O(log(q))
10
11 template<typename T> struct color {
12     set<tuple<int, int, T>> se;
13
14     vector<tuple<int, int, T>> update(int l, int r, T
15         val) {
16         auto it = se.upper_bound({r, INF, val});
17         if (it != se.begin() and get<1>(*prev(it)) >
18             r) {
19             auto [L, R, V] = *--it;
20             se.erase(it);
21             se.emplace(L, r, V), se.emplace(r+1, R, V
22         );
23         }
24         it = se.lower_bound({l, -INF, val});
25         if (it != se.begin() and get<1>(*prev(it)) >=
26             l) {
27             auto [L, R, V] = *--it;
28             se.erase(it);
29             se.emplace(L, l-1, V), it = se.emplace(l,
30             R, V).first;
31         }
32         vector<tuple<int, int, T>> ret;
33         for (; it != se.end() and get<0>(*it) <= r;
34             it = se.erase(it))
35             ret.push_back(*it);
36         se.emplace(l, r, val);
37         return ret;
38     }
39
40     T query(int i) {
41         auto it = se.upper_bound({i, INF, T()});
42         if (it == se.begin() or get<1>(*--it) < i)
43             return -1; // nao tem
44         return get<2>(*it);
45     }
46 };

```

1.13 Maxqueue

```

1 struct MaxQueue {
2     stack< pair<ll, ll> > in, out;

```

```

3
4 void add(ll x){
5     if(in.size())
6         in.push( { x, max(x, in.top().ss) } );
7     else
8         in.push( {x, x} );
9 }
10
11 ll get_max(){
12     if(in.size() > 0 && out.size() > 0)
13         return max(in.top().ss, out.top().ss);
14     else if(in.size() > 0) return in.top().ss;
15     else if(out.size() > 0) return out.top().ss;
16     else return INF;
17 }
18
19 void rem(){
20
21     if(out.size() == 0){
22         while(in.size()){
23             ll temp = in.top().ff, ma;
24             if(out.size() == 0) ma = temp;
25             else ma = max(temp, out.top().ss);
26             out.push({temp, ma});
27             in.pop();
28         }
29     }
30     //removendo o topo de out
31     out.pop();
32 }
33
34 }
35
36 ll size(){
37     return in.size() + out.size();
38 }
39
40 };

```

1.14 Triexor

```

1 struct Trie {
2
3     int nxt = 1, sz, maxLet = 2;
4     vector< vector<int> > trie;
5     vector<int> finish, paths;
6
7     Trie(int n){
8         sz = n;
9         trie.assign(sz + 10, vector<int>(maxLet,0));
10        finish.resize(sz + 10);
11        paths.resize(sz+10);
12    }
13
14    void add(int x){
15        int cur = 0;
16        for(int i = 31; i >= 0; i--){
17            int b = ( (x & (1 << i)) > 0);
18            if(trie[cur][b] == 0)
19                trie[cur][b] = nxt++;
20            cur = trie[cur][b];
21            paths[cur]++;
22        }
23        paths[cur]++;
24    }
25
26    void rem(int x){
27        int cur = 0;
28        for(int i = 31; i >= 0; i--){
29            int b = ( (x & (1 << i)) > 0);
30            cur = trie[cur][b];
31            paths[cur]--;
32        }

```

```

33        finish[cur]--;
34        paths[cur]--;
35    }
36
37    int query(int x){ //return the max xor with x
38        int ans = 0, cur = 0;
39
40        for(int i = 31; i >= 0; i--){
41            int b = ( (x & (1 << i)) > 0);
42            int bz = trie[cur][0];
43            int bo = trie[cur][1];
44
45            if(bz > 0 && bo > 0 && paths[bz] > 0 &&
46                paths[bo] > 0){
47                //cout << "Optimal" << endl;
48                cur = trie[cur][b ^ 1];
49                ans += (1 << i);
50            } else if(bz > 0 && paths[bz] > 0){
51                //cout << "Zero" << endl;
52                cur = trie[cur][0];
53                if(b) ans += (1 << i);
54            } else if(bo > 0 && paths[bo] > 0){
55                //cout << "One" << endl;
56                cur = trie[cur][1];
57                if(!b) ans += (1 << i);
58            } else {
59                break;
60            }
61        }
62        return ans;
63    }
64
65 };

```

1.15 Bit2d

```

1 struct BIT2D {
2
3     int n, m;
4     vector<vector<int>> bit;
5
6     BIT2D(int nn, int mm) {
7         //use as 0-indexed, but inside here I will
8         //use 1-indexed positions
9         n = nn + 2;
10        m = mm + 2;
11        bit.assign(n, vector<int>(m));
12    }
13
14    void update(int x, int y, int p) {
15        x++; y++;
16        assert(x > 0 && y > 0 && x <= n && y <= m);
17        for(; x < n; x += (x & (-x)))
18            for(int j = y; j < m; j += (j & (-j)))
19                bit[x][j] += p;
20    }
21
22    int sum(int x, int y) {
23        int ans = 0;
24        for(; x > 0; x -= (x & (-x)))
25            for(int j = y; j > 0; j -= (j & (-j)))
26                ans += bit[x][j];
27        return ans;
28    }
29
30    int query(int x, int y, int p, int q) {
31        //x...p on line, y...q on column
32        //sum from [x][y] to [p][q];
33        x++; y++; p++; q++;
34        assert(x > 0 && y > 0 && x <= n && y <= m);
35        assert(p > 0 && q > 0 && p <= n && q <= m);

```

```

35         return sum(p, q) - sum(x - 1, q) - sum(p, y - 1) + sum(x - 1, y - 1);
36     }
37
38 };
39

```

1.16 Treap Cp

```

1  mt19937 rng((int) chrono::steady_clock::now().
   time_since_epoch().count());
2
3  typedef struct item * pitem;
4
5  struct item {
6      int prior, value, cnt;
7      bool rev;
8      pitem l, r;
9
10     // Construtor para inicializar um nó com um
   valor dado
11     item(int _val) {
12         prior = rng();
13         value = _val;
14         cnt = 1; // Inicializa o contador como 1
15         rev = false; // Define o reverso como falso
   por padrão
16         l = r = nullptr;
17     }
18 };
19
20 int cnt (pitem it) {
21     return it ? it->cnt : 0;
22 }
23
24 void upd_cnt (pitem it) {
25     if (it)
26         it->cnt = cnt(it->l) + cnt(it->r) + 1;
27 }
28
29 void push (pitem it) {
30     if (it && it->rev) {
31         it->rev = false;
32         swap (it->l, it->r);
33         if (it->l) it->l->rev ^= true;
34         if (it->r) it->r->rev ^= true;
35     }
36 }
37
38 void merge (pitem & t, pitem l, pitem r) {
39     push (l);
40     push (r);
41     if (!l || !r)
42         t = l ? l : r;
43     else if (l->prior > r->prior)
44         merge (l->r, l->r, r), t = l;
45     else
46         merge (r->l, l, r->l), t = r;
47     upd_cnt (t);
48 }
49
50 // essa func quebra um range baseado na key e salva
   as duas partes em l, r
51 void split (pitem t, pitem & l, pitem & r, int key,
   int add = 0) {
52     if (!t)
53         return void( l = r = 0 );
54     push (t);
55     int cur_key = add + cnt(t->l);
56     if (key <= cur_key)
57         split (t->l, l, t->l, key, add), r = t;
58     else

```

```

   split (t->r, t->r, r, key, add + 1 + cnt(t->l
   )), l = t;
60     upd_cnt (t);
61 }
62
63 // essa inverte o range l, r do nó t
64 void reverse (pitem t, int l, int r) {
65     pitem t1, t2, t3;
66     split (t, t1, t2, l);
67     split (t2, t2, t3, r-l+1);
68     t2->rev ^= true;
69     merge (t, t1, t2);
70     merge (t, t, t3);
71 }
72
73 vector<int> ans;
74
75 void output (pitem t) {
76     if (!t) return;
77     push (t);
78     output (t->l);
79     // pode printar o valor direto aq tmb
80     ans.push_back(t->value);
81     output (t->r);
82 }
83
84 // https://cses.fi/problemset/task/2072/
85 // cortar o range [l, r] e cola no final
86 void cut_and_paste(pitem root, int l, int r) {
87     pitem A, B, C, D;
88     // separa a root em caras com indice < l e >= l
89     // e salva as partes em A, B
90     split(root, A, B, l);
91     // pega a parte B (indices i >= l) e pega
92     // exatamente o tamanho que vc quer
93     // salva as partes em C e D
94     split(B, C, D, r - l + 1);
95     // Da merge dos indices i < l com a parte i > r
96     merge(root, A, D);
97     // da merge do pedaço que vc queria final e
98     // deixa salvo em root
99     merge(root, root, C);
100
101 void solve() {
102
103     int n, q;
104     cin >> n >> q;
105
106     string s;
107     cin >> s;
108
109     pitem root = nullptr;
110
111     for(int i = 0; i < n; i++) {
112         pitem newNode = new item(i);
113         merge(root, root, newNode);
114     }
115
116     while(q--) {
117         int l, r;
118         cin >> l >> r;
119         cut_and_paste(root, l - 1, r - 1);
120     }
121
122     output(root);
123
124     for(int i = 0; i < n; i++) {
125         cout << s[ans[i]];
126     }
127
128     cout << "\n";
129

```

1.17 Querytree

```

130 }

1 struct QueryTree {
2     int n, t = 0, l = 3, build = 0, euler = 0;
3     vector<ll> dist;
4     vector<int> in, out, d;
5     vector<vector<int>> sobe;
6     vector<vector<pair<int, ll>>> arr;
7     vector<vector<ll>> table_max; //max edge
8     vector<vector<ll>> table_min; //min edge
9
10    QueryTree(int nn) {
11        n = nn + 5;
12        arr.resize(n);
13        in.resize(n);
14        out.resize(n);
15        d.resize(n);
16        dist.resize(n);
17        while( (1 << l) < n ) l++;
18        sobe.assign(n + 5, vector<int>(>+1));
19        table_max.assign(n + 5, vector<ll>(1));
20        table_min.assign(n + 5, vector<ll>(1));
21    }
22
23    void add_edge(int u, int v, ll w){ //
24        bidirectional edge with weight w
25        arr[u].push_back({v, w});
26        arr[v].push_back({u, w});
27    }
28
29    //assert the root of tree is node 1 or change the
30    'last' in the next function
31    void Euler_Tour(int u, int last = 1, ll we = 0,
32    int depth = 0, ll sum = 0){ //euler tour
33    euler = 1; //remember to use this function
34    before the queries
35    in[u] = t++;
36    d[u] = depth;
37    dist[u] = sum; //sum = sum of the values in
38    edges from root to node u
39    sobe[u][0] = last; //parent of u. parent of 1
40    is 1
41    table_max[u][0] = we;
42    table_min[u][0] = we;
43    for(auto v: arr[u]) if(v.ff != last){
44        Euler_Tour(v.ff, u, v.ss, depth + 1, sum
45        + v.ss);
46    }
47    out[u] = t++;
48    }
49
50    void build_table(){ //binary lifting
51    assert(euler);
52    build = 1; //remeber use this function before
53    queries
54    for(int k = 1; k < l; k++){
55        for(int i = 1; i <= n; i++){
56            sobe[i][k] = sobe[sobe[i][k-1]][k-1];
57            table_max[i][k] = max(table_max[i][k-1],
58            table_max[sobe[i][k-1]][k-1]);
59            table_min[i][k] = min(table_min[i][k-1],
60            table_min[sobe[i][k-1]][k-1]);
61        }
62    }
63
64    int is_ancestor(int u, int v){ // return 1 if u
65    is ancestor of v
66    assert(euler);
67    return in[u] <= in[v] && out[u] >= out[v];
68    }
69
70    int lca(int u, int v){ //return lca of u and v
71    assert(build && euler);
72    if(is_ancestor(u,v)) return u;
73    if(is_ancestor(v,u)) return v;
74    int lca = u;
75    for(int k = l - 1; k >= 0; k--){
76        int tmp = sobe[lca][k];
77        if(!is_ancestor(tmp, v)){
78            lca = tmp;
79        }
80    }
81    return sobe[lca][0];
82    }
83
84    int lca(int u, int v, int root) { //return lca of
85    u and v when tree is rooted at 'root'
86    return lca(u, v) ^ lca(v, root) ^ lca(root, u
87    ); //magic
88    }
89
90    int up_k(int u, int qt){ //return node k levels
91    higher starting from u
92    assert(build && euler);
93    for(int b = 0; b < l; b++){
94        if(qt%2) u = sobe[u][b];
95        qt >>= 1;
96    }
97    return u;
98    }
99
100    ll goUpMax(int u, int to){ //return the max
101    weigth of a edge going from u to 'to'
102    assert(build);
103    if(u == to) return 0;
104    ll mx = table_max[u][0];
105    for(int k = l - 1; k >= 0; k--){
106        int tmp = sobe[u][k];
107        if( !is_ancestor(tmp, to) ){
108            mx = max(mx, table_max[u][k]);
109            u = tmp;
110        }
111    }
112    return max(mx, table_max[u][0]);
113    }
114
115    ll max_edge(int u, int v){ //return the max
116    weight of a edge in the simple path from u to v
117    assert(build);
118    int ancestor = lca(u, v);
119    ll a = goUpMax(u, ancestor), b = goUpMax(v,
120    ancestor);
121    if(ancestor == u) return b;
122    else if(ancestor == v) return a;
123    return max(a,b);
124    }
125
126    ll goUpMin(int u, int to){ //return the min
127    weight of a edge going from u to 'to'
128    assert(build);
129    if(u == to) return oo;
130    ll mx = table_min[u][0];
131    for(int k = l - 1; k >= 0; k--){
132        int tmp = sobe[u][k];
133        if( !is_ancestor(tmp, to) ){
134            mx = min(mx, table_min[u][k]);
135            u = tmp;
136        }
137    }
138    return min(mx, table_min[u][0]);
139    }
140
141    ll min_edge(int u, int v){ //return the min

```

```

125 weight of a edge in the simple path from u to v
126     assert(build);
127     int ancestor = lca(u, v);
128     ll a = goUpMin(u, ancestor), b = goUpMin(v,
129 ancestor);
130     if(ancestor == u) return b;
131     else if(ancestor == v) return a;
132     return min(a,b);
133 }
134
135 ll query_dist(int u, int v){ //distance of nodes
136 u and v
137     int x = lca(u, v);
138     return dist[u] - dist[x] + dist[v] - dist[x];
139 }
140
141 int kth_between(int u, int v, int k){ //kth node
142 in the simple path from u to v; if k = 1, ans = u
143 k--;
144     int x = lca(u, v);
145     if( k > d[u] - d[x] ){
146         k -= (d[u] - d[x]);
147         return up_k(v, d[v]-d[x]-k);
148     }
149     return up_k(u, k);
150 }
151
152 };
153
154 int main() {
155     ios::sync_with_stdio(false);
156     cin.tie(NULL);
157
158     int t = 1, n, u, v, w, k;
159     string s;
160     cin >> t;
161     while(t--){
162         cin >> n;
163         QueryTree arr(n);
164         for(int i = 1; i < n; i++){
165             cin >> u >> v >> w;
166             arr.add_edge(u,v,w);
167         }
168         arr.Euler_Tour(1);
169         arr.build_table();
170         while(cin >> s, s != "DONE"){
171             cin >> u >> v;
172             if(s == "DIST") {
173                 cout << arr.query_dist(u, v) << "\n";
174             } else {
175                 cin >> k;
176                 cout << arr.kth_between(u,v,k) << "\n";
177             }
178         }
179     }
180 }

```

1.18 Sparse

```

1 struct Sparse {
2
3     vector<vector<int>> arr;
4
5     int op(int& a, int& b){ //min, max, gcd, lcm, and
6     , or
7         return min(a,b);
8         //return __gcd(a,b);
9         //return max(a,b);
10    }

```

```

11 Sparse(vector<int>& v){ //Constrói a tabela
12     int n = v.size(), logn = 0;
13     while((1<<logn) <= n) logn++;
14     arr.assign(n, vector<int>(logn, 0));
15     for(int i = 0; i < n; i++)
16         arr[i][0] = v[i];
17     for(int k = 1; k < logn; k++){
18         for(int i = 0; i < n; i++){
19             if(i + ( 1 << k) - 1 >= n)
20                 break;
21             int p = i+( 1 << (k-1) );
22             arr[i][k] = op( arr[i][ k-1 ] , arr[p
23 ] [k-1] );
24         }
25     }
26
27     int query(int l, int r){
28         int pot = 31 - __builtin_clz(r-l+1); //r-l+1
29         sãẽo INTEIROS, nãẽo ll
30         int k = (1 << pot) ;
31         return op( arr[l][pot] , arr[ r - (k-1) ][
32 pot] );
33     };

```

1.19 Mex

```

1 // Mex
2 //
3 // facilita queries de mex com update
4 //
5 // N eh o maior valor possível do mex
6 // add(x) = adiciona x
7 // rem(x) = remove x
8 //
9 // O(log N) por insert
10 // O(1) por query
11
12 struct Mex {
13     map<int, int> cnt;
14     set<int> possible;
15
16     Mex(int n) {
17         for (int i = 0; i <= n + 1; i++) {
18             possible.insert(i);
19         }
20     }
21
22     void add(int x) {
23         cnt[x]++;
24         possible.erase(x);
25     }
26
27     void rem(int x) {
28         cnt[x]--;
29
30         if (cnt[x] == 0) {
31             possible.insert(x);
32         }
33     }
34
35     int query() {
36         return *(possible.begin());
37     }
38 };

```

1.20 Cht

```

1 // CHT (tiagodfs)
2

```

```

3  const ll is_query = -LLINF;
4  struct Line{
5      ll m, b;
6      mutable function<const Line*> succ;
7      bool operator<(const Line& rhs) const{
8          if(rhs.b != is_query) return m < rhs.m;
9          const Line* s = succ();
10         if(!s) return 0;
11         ll x = rhs.m;
12         return b - s->b < (s->m - m) * x;
13     }
14 };
15 struct Cht : public multiset<Line>{ // maintain max m
16     *x+b
17     bool bad(iterator y){
18         auto z = next(y);
19         if(y == begin()){
20             if(z == end()) return 0;
21             return y->m == z->m && y->b <= z->b;
22         }
23         auto x = prev(y);
24         if(z == end()) return y->m == x->m && y->b <=
25         x->b;
26         return (ld)(x->b - y->b)*(z->m - y->m) >= (ld)
27         )(y->b - z->b)*(y->m - x->m);
28     }
29     void insert_line(ll m, ll b){ // min -> insert (-
30         m, -b) -> -eval()
31         auto y = insert({ m, b });
32         y->succ = [=]{ return next(y) == end() ? 0 :
33         &*next(y); };
34         if(bad(y)){ erase(y); return; }
35         while(next(y) != end() && bad(next(y))) erase
36         (next(y));
37         while(y != begin() && bad(prev(y))) erase(
38         prev(y));
39     }
40     ll eval(ll x){
41         auto l = *lower_bound((Line) { x, is_query });
42         ;
43         return l.m * x + l.b;
44     }
45 };

```

1.21 Ordered Set

```

1  // Ordered Set
2  //
3  // set roubado com mais operacoes
4  //
5  // para alterar para multiset
6  // trocar less para less_equal
7  //
8  // ordered_set<int> s
9  //
10 // order_of_key(k) // number of items strictly
11 // smaller than k -> int
12 // find_by_order(k) // k-th element in a set (
13 // counting from zero) -> iterator
14 //
15 // https://cses.fi/problemset/task/2169
16 //
17 // O(log N) para insert, erase (com iterator),
18 // order_of_key, find_by_order
19
20 using namespace __gnu_pbds;
21 template <typename T>
22 using ordered_set = tree<T, null_type, less<T>,
23 rb_tree_tag, tree_order_statistics_node_update>;
24
25 void erase(ordered_set& a, int x){
26     int r = a.order_of_key(x);
27     auto it = a.find_by_order(r);

```

```

24     a.erase(it);
25 }

```

1.22 Manhattan Mst

```

1  /**
2   * Author: chilli, Takanori MAEHARA
3   * Date: 2019-11-02
4   * License: CCO
5   * Source: https://github.com/spaghetti-source/
6   * algorithm/blob/master/geometry/rectilinear_mst.cc
7   * Description: Given N points, returns up to 4*N
8   * edges, which are guaranteed
9   * to contain a minimum spanning tree for the graph
10  * with edge weights w(p, q) =
11  * |p.x - q.x| + |p.y - q.y|. Edges are in the form (
12  * distance, src, dst). Use a
13  * standard MST algorithm on the result to find the
14  * final MST.
15  * Time: O(N \log N)
16  * Status: Stress-tested
17  */
18 /**
19  * Author: Ulf Lundstrom
20  * Date: 2009-02-26
21  * License: CCO
22  * Source: My head with inspiration from tinyKACTL
23  * Description: Class to handle points in the plane.
24  * T can be e.g. double or long long. (Avoid int.)
25  * Status: Works fine, used a lot
26  */
27 #pragma once
28
29 template <class T> int sgn(T x) { return (x > 0) - (x
30 < 0); }
31
32 template<class T>
33 struct Point {
34     typedef Point P;
35     T x, y;
36     explicit Point(T x=0, T y=0) : x(x), y(y) {}
37     bool operator<(P p) const { return tie(x,y) < tie
38     (p.x,p.y); }
39     bool operator==(P p) const { return tie(x,y)==tie
40     (p.x,p.y); }
41     P operator+(P p) const { return P(x+p.x, y+p.y);
42     }
43     P operator-(P p) const { return P(x-p.x, y-p.y);
44     }
45     P operator*(T d) const { return P(x*d, y*d); }
46     P operator/(T d) const { return P(x/d, y/d); }
47     T dot(P p) const { return x*p.x + y*p.y; }
48     T cross(P p) const { return x*p.y - y*p.x; }
49     T cross(P a, P b) const { return (a-*this).cross(
50     b-*this); }
51     T dist2() const { return x*x + y*y; }
52     double dist() const { return sqrt((double)dist2()
53     ); }
54     // angle to x-axis in interval [-pi, pi]
55     double angle() const { return atan2(y, x); }
56     P unit() const { return *this/dist(); } // makes
57     dist()=1
58     P perp() const { return P(-y, x); } // rotates
59     +90 degrees
60     P normal() const { return perp().unit(); }
61     // returns point rotated 'a' radians ccw around
62     the origin
63     P rotate(double a) const {
64         return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a))
65     }; }
66
67 friend ostream& operator<<(ostream& os, P p) {
68     return os << "(" << p.x << "," << p.y << ")";
69 }

```

```

52 };
53
54 typedef Point<int> P;
55 vector<array<int, 3>> manhattanMST(vector<P> ps) {
56     vi id(sz(ps));
57     iota(all(id), 0);
58     vector<array<int, 3>> edges;
59     rep(k,0,4) {
60         sort(all(id), [&](int i, int j) {
61             return (ps[i]-ps[j]).x < (ps[j]-ps[i]).y
62         });
63         map<int, int> sweep;
64         for (int i : id) {
65             for (auto it = sweep.lower_bound(-ps[i].y
66             );
67                 it != sweep.end(); sweep.
68                 erase(it++)) {
69                 int j = it->second;
70                 P d = ps[i] - ps[j];
71                 if (d.y > d.x) break;
72                 edges.push_back({d.y + d.x, i, j});
73             }
74             sweep[-ps[i].y] = i;
75         }
76         for (P& p : ps) if (k & 1) p.x = -p.x; else
77         swap(p.x, p.y);
78     }
79     return edges;
80 }

```

1.23 Bit

```

1 struct BIT {
2     int n, LOGN = 0;
3     vector<ll> bit;
4
5     BIT(int nn){
6         n = nn + 10;
7         bit.resize(n + 10, 0);
8         while( (1LL << LOGN) <= n ) LOGN++;
9     }
10
11     ll query(int x){
12         x++;
13         ll ans = 0;
14         while(x > 0){
15             ans += bit[x];
16             x -= (x & (-x));
17         }
18         return ans;
19     }
20
21     void update(int x, ll val){
22         x++;
23         while(x < (int)bit.size()){
24             bit[x] += val;
25             x += (x & (-x));
26         }
27     }
28
29     int findkth(int k){
30         //kth smallest, 0(logN)
31         //use position i to count how many times
32         value 'i' appear
33         int sum = 0, pos = 0;
34         for(int i = LOGN; i >= 0; i--){
35             if(pos + (1LL << i) < n && sum + bit[pos
36             + (1LL << i)] < k){
37                 sum += bit[pos + (1LL << i)];
38                 pos += (1LL << i);
39             }
40         }
41         return pos;

```

```

40     }
41     /*
42     int findkth(int k){
43         //kth smallest, 0(log^2(N))
44         //use position i to count how many times
45         value 'i' appear
46         int x = 0, mx = 200;
47         for(int b = n; b > 0 && mx > 0; b /= 2){
48             while( x+b < n && query(x+b) < k && mx--
49             > 0 ){
50                 x += b;
51             }
52             return x+1;
53         }
54     };

```

1.24 Psum2d

```

1 struct PSum {
2
3     vector<vi> arr;
4     int n, m, initialized = 0;
5
6     PSum(int _n, int _m) {
7         n = _n;
8         m = _m;
9         arr.resize(n + 2);
10        arr.assign(n + 2, vector<int>(m + 2, 0));
11    }
12
13    void add(int a, int b, int c) {
14        //a and b are 0-indexed
15        arr[a + 1][b + 1] += c;
16    }
17
18    void init() {
19        for(int i = 1; i <= n; i++) {
20            for(int j = 1; j <= m; j++) {
21                arr[i][j] += arr[i][j - 1];
22                arr[i][j] += arr[i - 1][j];
23                arr[i][j] -= arr[i - 1][j - 1];
24            }
25        }
26        initialized = 1;
27    }
28
29    int query(int a, int b, int c, int d) {
30        // sum of a...c and b...d
31        // a, b, c and d are 0-indexed
32        assert(initialized);
33        return arr[c + 1][d + 1] - arr[a][d + 1] -
34        arr[c + 1][b] + arr[a][b];
35    }
36 };

```

1.25 Kruskal

```

1 struct Edge {
2     int u, v;
3     ll weight;
4
5     Edge() {}
6
7     Edge(int u, int v, ll weight) : u(u), v(v),
8     weight(weight) {}
9
10    bool operator<(Edge const& other) {
11        return weight < other.weight;

```

```

12 };
13
14 vector<Edge> kruskal(vector<Edge> edges, int n) {
15     vector<Edge> result;
16     ll cost = 0;
17
18     sort(edges.begin(), edges.end());
19     DSU dsu(n);
20
21     for (auto e : edges) {
22         if (!dsu.same(e.u, e.v)) {
23             cost += e.weight;
24             result.push_back(e);
25             dsu.unite(e.u, e.v);
26         }
27     }
28
29     return result;
30 }

```

2 String

2.1 Suffix Array

```

1 // Credits to Brunomaletta
2 // https://github.com/brunomaletta/Biblioteca/blob/
3 // master/Codigo/Strings/suffixArray2.cpp
4 // Suffix Array - O(n log n)
5 //
6 // kasai recebe o suffix array e calcula lcp[i],
7 // o lcp entre s[sa[i],...,n-1] e s[sa[i+1],...,n-1]
8 //
9 // Complexidades:
10 // suffix_array - O(n log(n))
11 // kasai - O(n)
12
13 vector<int> suffix_array(string s) {
14     s += "$";
15     int n = s.size(), N = max(n, 260);
16     vector<int> sa(n), ra(n);
17     for(int i = 0; i < n; i++) sa[i] = i, ra[i] = s[i];
18
19     for(int k = 0; k < n; k ? k *= 2 : k++) {
20         vector<int> nsa(sa), nra(n), cnt(N);
21
22         for(int i = 0; i < n; i++) nsa[i] = (nsa[i]-k+n)%n, cnt[ra[i]]++;
23         for(int i = 1; i < N; i++) cnt[i] += cnt[i-1];
24         for(int i = n-1; i+1; i--) sa[--cnt[ra[nsa[i]]]] = nsa[i];
25
26         for(int i = 1, r = 0; i < n; i++) nra[sa[i]] = r += ra[sa[i]] != ra[sa[i-1]] or ra[(sa[i]+k)%n] != ra[(sa[i-1]+k)%n];
27         ra = nra;
28         if (ra[sa[n-1]] == n-1) break;
29     }
30     return vector<int>(sa.begin()+1, sa.end());
31 }
32
33 vector<int> kasai(string s, vector<int> sa) {
34     int n = s.size(), k = 0;
35     vector<int> ra(n), lcp(n);
36     for (int i = 0; i < n; i++) ra[sa[i]] = i;
37
38     for (int i = 0; i < n; i++, k -= !!k) {
39         if (ra[i] == n-1) { k = 0; continue; }
40         int j = sa[ra[i]+1];

```

```

42         while (i+k < n and j+k < n and s[i+k] == s[j+k]) k++;
43         lcp[ra[i]] = k;
44     }
45     return lcp;
46 }

```

2.2 Hash

```

1 struct Hash {
2     ll MOD, P;
3     int n; string s;
4     vector<ll> h, hi, p;
5     Hash() {}
6     Hash(string s, ll MOD, ll P = 31): s(s), MOD(MOD), P(P), n(s.size()), h(n), hi(n), p(n) {
7         for (int i=0; i<n; i++) p[i] = (i ? P*p[i-1]:1) % MOD;
8         for (int i=0; i<n; i++)
9             h[i] = (s[i] + (i ? h[i-1]:0) * P) % MOD;
10        for (int i=n-1; i>=0; i--)
11            hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * P) % MOD;
12    }
13    int query(int l, int r) {
14        ll hash = (h[r] - (l ? h[l-1]*p[r-l+1]:0)) % MOD;
15        return hash < 0 ? hash + MOD : hash;
16    }
17    int query_inv(int l, int r) {
18        ll hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-l+1]:0)) % MOD;
19        return hash < 0 ? hash + MOD : hash;
20    }
21 };
22
23 struct DoubleHash {
24     const ll MOD1 = 90264469;
25     const ll MOD2 = 25699183;
26
27     Hash hash1, hash2;
28
29     DoubleHash();
30
31     DoubleHash(string s) : hash1(s, MOD1), hash2(s, MOD2) {}
32
33     pair<int, int> query(int l, int r) {
34         return { hash1.query(l, r), hash2.query(l, r) };
35     }
36
37     pair<int, int> query_inv(int l, int r) {
38         return { hash1.query_inv(l, r), hash2.query_inv(l, r) };
39     }
40 };
41
42 struct TripleHash {
43     const ll MOD1 = 90264469;
44     const ll MOD2 = 25699183;
45     const ll MOD3 = 81249169;
46
47     Hash hash1, hash2, hash3;
48
49     TripleHash();
50
51     TripleHash(string s) : hash1(s, MOD1), hash2(s, MOD2), hash3(s, MOD3) {}
52
53     tuple<int, int, int> query(int l, int r) {
54         return { hash1.query(l, r), hash2.query(l, r), hash3.query(l, r) };

```



```

55     }
56
57     tuple<int, int, int> query_inv(int l, int r) {
58         return { hash1.query_inv(l, r), hash2.
59             query_inv(l, r), hash3.query_inv(l, r) };
60     };
61
62     struct HashK {
63         vector<ll> primes; // more primes = more hashes
64         vector<Hash> hash;
65
66         HashK();
67
68         HashK(string s, vector<ll> primes): primes(primes)
69         {
70             for (auto p : primes) {
71                 hash.push_back(Hash(s, p));
72             }
73
74             vector<int> query(int l, int r) {
75                 vector<int> ans;
76
77                 for (auto h : hash) {
78                     ans.push_back(h.query(l, r));
79                 }
80
81                 return ans;
82             }
83
84             vector<int> query_inv(int l, int r) {
85                 vector<int> ans;
86
87                 for (auto h : hash) {
88                     ans.push_back(h.query_inv(l, r));
89                 }
90
91                 return ans;
92             }
93     };

```

2.3 Split

```

1 vector<string> split(string s, char key=' ') {
2     vector<string> ans;
3     string aux = "";
4
5     for (int i = 0; i < (int)s.size(); i++) {
6         if (s[i] == key) {
7             if (aux.size() > 0) {
8                 ans.push_back(aux);
9                 aux = "";
10            }
11        } else {
12            aux += s[i];
13        }
14    }
15
16    if ((int)aux.size() > 0) {
17        ans.push_back(aux);
18    }
19
20    return ans;
21 }

```

2.4 Trie Xor

```

1 // TrieXOR
2 //
3 // adiciona, remove e verifica se existe strings
  binarias

```

```

4 // max_xor(x) = maximiza o xor de x com algum valor
  da trie
5 //
6 // raiz = 0
7 //
8 // https://codeforces.com/problemset/problem/706/D
9 //
10 // 0(|s|) adicionar, remover e buscar
11
12 struct TrieXOR {
13     int n, alph_sz, nxt;
14     vector<vector<int>> trie;
15     vector<int> finish, paths;
16
17     TrieXOR() {}
18
19     TrieXOR(int n, int alph_sz = 2) : n(n), alph_sz(
20         alph_sz) {
21         nxt = 1;
22         trie.assign(n, vector<int>(alph_sz));
23         finish.assign(n * alph_sz, 0);
24         paths.assign(n * alph_sz, 0);
25     }
26
27     void add(int x) {
28         int curr = 0;
29
30         for (int i = 31; i >= 0; i--) {
31             int b = ((x >> i) & 1);
32
33             if (trie[curr][b] == 0)
34                 trie[curr][b] = nxt++;
35
36             paths[curr]++;
37             curr = trie[curr][b];
38         }
39
40         paths[curr]++;
41         finish[curr]++;
42     }
43
44     void rem(int x) {
45         int curr = 0;
46
47         for (int i = 31; i >= 0; i--) {
48             int b = ((x >> i) & 1);
49
50             paths[curr]--;
51             curr = trie[curr][b];
52         }
53
54         paths[curr]--;
55         finish[curr]--;
56     }
57
58     int search(int x) {
59         int curr = 0;
60
61         for (int i = 31; i >= 0; i--) {
62             int b = ((x >> i) & 1);
63
64             if (trie[curr][b] == 0) return false;
65
66             curr = trie[curr][b];
67         }
68
69         return (finish[curr] > 0);
70     }
71
72     int max_xor(int x) { // maximum xor with x and
73         any number of trie
74         int curr = 0, ans = 0;

```

```

74     for (int i = 31; i >= 0; i--) {
75         int b = ((x & (1 << i)) > 0);
76         int want = b ^ 1;
77
78         if (trie[curr][want] == 0 || paths[trie[
curr][want]] == 0) want ^= 1;
79         if (trie[curr][want] == 0 || paths[trie[
curr][want]] == 0) break;
80         if (want != b) ans |= (1 << i);
81
82         curr = trie[curr][want];
83     }
84
85     return ans;
86 }
87 };

```

2.5 Prefix Func

```

1
2 // Credits to cp algo
3 // pi[i] is the length of the longest proper prefix
  of the substring
4 // s[0...i] is also a suffix of this
  substring
5 // abcbabcd -> 0001230 and aabaaab -> 0101223
6 // pi[0] = 0
7 vector<int> prefix_function(string s) {
8     int n = (int)s.length();
9     vector<int> pi(n);
10    for (int i = 1; i < n; i++) {
11        int j = pi[i-1];
12        while (j > 0 && s[i] != s[j])
13            j = pi[j-1];
14        if (s[i] == s[j])
15            j++;
16        pi[i] = j;
17    }
18    return pi;
19 }

```

2.6 Is Substring

```

1 // equivalente ao in do python
2
3 bool is_substring(string a, string b){ // verifica se
  a é substring de b
4     for(int i = 0; i < b.size(); i++){
5         int it = i, jt = 0; // b[it], a[jt]
6
7         while(it < b.size() && jt < a.size()){
8             if(b[it] != a[jt])
9                 break;
10
11             it++;
12             jt++;
13
14             if(jt == a.size())
15                 return true;
16         }
17     }
18
19     return false;
20 }

```

3 Math

3.1 Fexp

```

1 using ll = long long;
2

```

```

3 ll fexp(ll base, ll exp, ll m) {
4     ll ans = 1;
5     base %= m;
6
7     while (exp > 0) {
8         if (exp % 2 == 1) {
9             ans = (ans * base) % m;
10        }
11
12        base = (base * base) % m;
13        exp /= 2;
14    }
15
16    return ans;
17 }

```

3.2 Sieve

```

1 vector<int> sieve(int MAXN){
2     //list of prime numbers up to MAXN
3     vector<int> primes;
4     bitset<(int)1e7> not_prime;
5     not_prime[0] = 1;
6     not_prime[1] = 1;
7     for(int i = 2; i <= MAXN; i++){
8         if(!not_prime[i]){
9             primes.push_back(i);
10            for(ll j = 1LL * i * i; j <= MAXN; j += i
11                )
12                not_prime[(int)j] = 1;
13        }
14    }
15    return primes;
16 }

```

3.3 Ifac

```

1 // inverse of factorial
2
3 mint fac[N], ifac[N];
4
5 void build_fac() {
6     fac[0] = 1;
7
8     for (int i = 1; i < N; i++) {
9         fac[i] = fac[i - 1] * i;
10    }
11
12    ifac[N - 1] = inv(fac[N - 1]);
13
14    for (int i = N - 2; i >= 0; i--) {
15        ifac[i] = ifac[i + 1] * (i + 1);
16    }
17 }

```

3.4 Division Trick

```

1 for(int l = 1, r; l <= n; l = r + 1) {
2     r = n / (n / l);
3     // n / x yields the same value for l <= x <= r
4 }
5 for(int l, r = n; r > 0; r = l - 1) {
6     int tmp = (n + r - 1) / r;
7     l = (n + tmp - 1) / tmp;
8     // (n+x-1) / x yields the same value for l <= x
  <= r
9 }

```

3.5 Factorization

```

1 // nson
2
3 using ll = long long;
4
5 vector<pair<ll, int>> factorization(ll n) {
6     vector<pair<ll, int>> ans;
7
8     for (ll p = 2; p*p <= n; p++) {
9         if (n%p == 0) {
10             int expoente = 0;
11
12             while (n%p == 0) {
13                 n /= p;
14                 expoente++;
15             }
16
17             ans.push_back({p, expoente});
18         }
19     }
20
21     if (n > 1) {
22         ans.push_back({n, 1});
23     }
24
25     return ans;
26 }

```

3.6 Fft Quirino

```

1 // FFT
2 //
3 // boa em memÃria e ok em tempo
4 //
5 // https://codeforces.com/group/YgJmumGtHD/contest/528947/problem/H (maratona mineira)
6
7 using cd = complex<double>;
8 const double PI = acos(-1);
9
10 void fft(vector<cd> &A, bool invert) {
11     int N = size(A);
12
13     for (int i = 1, j = 0; i < N; i++) {
14         int bit = N >> 1;
15         for (; j & bit; bit >>= 1)
16             j ^= bit;
17         j ^= bit;
18
19         if (i < j)
20             swap(A[i], A[j]);
21     }
22
23     for (int len = 2; len <= N; len <= 1) {
24         double ang = 2 * PI / len * (invert ? -1 : 1);
25         cd wlen(cos(ang), sin(ang));
26         for (int i = 0; i < N; i += len) {
27             cd w(1);
28             for (int j = 0; j < len/2; j++) {
29                 cd u = A[i+j], v = A[i+j+len/2] * w;
30                 A[i+j] = u + v;
31                 A[i+j+len/2] = u - v;
32                 w *= wlen;
33             }
34         }
35     }
36
37     if (invert) {
38         for (auto &x : A)
39             x /= N;
40     }
41 }
42

```

```

43 vector<int> multiply(vector<int> const& A, vector<int>
44 > const& B) {
45     vector<cd> fa(begin(A), end(A)), fb(begin(B), end(B));
46
47     int N = 1;
48     while (N < size(A) + size(B))
49         N <= 1;
50     fa.resize(N);
51     fb.resize(N);
52
53     fft(fa, false);
54     fft(fb, false);
55     for (int i = 0; i < N; i++)
56         fa[i] *= fb[i];
57     fft(fa, true);
58
59     vector<int> result(N);
60     for (int i = 0; i < N; i++)
61         result[i] = round(fa[i].real());
62     return result;
63 }

```

3.7 Divisors

```

1 vector<ll> divisors(ll n) {
2     vector<ll> ans;
3
4     for (ll i = 1; i*i <= n; i++) {
5         if (n%i == 0) {
6             ll value = n/i;
7
8             ans.push_back(i);
9             if (value != i) {
10                 ans.push_back(value);
11             }
12         }
13     }
14
15     return ans;
16 }

```

3.8 Number Sum Product Of Divisors

```

1 // CSES - Divisor Analysis
2 // Print the number, sum and product of the divisors.
3 // Since the input number may be large, it is given
4 // as a prime factorization.
5 //
6 // Input:
7 // The first line has an integer n: the number of
8 // parts in the prime factorization.
9 // After this, there are n lines that describe the
10 // factorization. Each line has two numbers x and k
11 // where x is a prime and k is its power.
12 //
13 // Output:
14 // Print three integers modulo 10^9+7: the number,
15 // sum and product of the divisors.
16 //
17 // Constraints:
18 // (1 <= n <= 1e5) ; (2 <= x <= 1e6) ; (1 <= k <= 1e9)
19 // ; each x is a distinct prime
20
21 #include <bits/stdc++.h>
22 typedef long long ll;
23 using namespace std;
24
25 const ll MOD = 1e9 + 7;
26
27 ll expo(ll base, ll pow) {
28     ll ans = 1;
29     while (pow) {
30

```

```

24         if (pow & 1) ans = ans * base % MOD;
25         base = base * base % MOD;
26         pow >>= 1;
27     }
28     return ans;
29 }
30
31 ll p[100001], k[100001];
32
33 int main() {
34     cin.tie(0) -> sync_with_stdio(0);
35     int n;
36     cin >> n;
37     for (int i = 0; i < n; i++) cin >> p[i] >> k[i];
38     ll div_cnt = 1, div_sum = 1, div_prod = 1,
39     div_cnt2 = 1;
40     for (int i = 0; i < n; i++) {
41         div_cnt = div_cnt * (k[i] + 1) % MOD;
42         div_sum = div_sum * (expo(p[i], k[i] + 1) -
43         1) % MOD *
44         expo(p[i] - 1, MOD - 2) % MOD;
45         div_prod = expo(div_prod, k[i] + 1) *
46         expo(expo(p[i], (k[i] * (k[i] + 1)
47         / 2)), div_cnt2) % MOD;
48         div_cnt2 = div_cnt2 * (k[i] + 1) % (MOD - 1);
49     }
50     cout << div_cnt << ' ' << div_sum << ' ' <<
51     div_prod;
52     return 0;
53 }

```

3.9 Is Prime

```

1 bool is_prime(ll n) {
2     if (n <= 1) return false;
3     if (n == 2) return true;
4
5     for (ll i = 2; i*i <= n; i++) {
6         if (n % i == 0)
7             return false;
8     }
9
10    return true;
11 }

```

3.10 Log Any Base

```

1 int intlog(double base, double x) {
2     return (int)(log(x) / log(base));
3 }

```

3.11 Ceil

```

1 using ll = long long;
2
3 // avoid overflow
4 ll division_ceil(ll a, ll b) {
5     return 1 + ((a - 1) / b); // if a != 0
6 }
7
8 int intceil(int a, int b) {
9     return (a+b-1)/b;
10 }

```

4 Primitives

4.1 Set Union Intersection

```

1 // Template pra fazer união e intercessão de sets
  de forma fácil

```

```

2 // Usar + para uniao e * para intercessão
3 // Source: https://stackoverflow.com/questions
  /13448064/how-to-find-the-intersection-of-two-stl
  -sets
4
5 template <class T, class CMP = std::less<T>, class
  ALLOC = std::allocator<T> >
6 std::set<T, CMP, ALLOC> operator * (
7     const std::set<T, CMP, ALLOC> &s1, const std::set<T
  , CMP, ALLOC> &s2)
8 {
9     std::set<T, CMP, ALLOC> s;
10    std::set_intersection(s1.begin(), s1.end(), s2.
11    begin(), s2.end(),
12    std::inserter(s, s.begin()));
13    return s;
14 }
15
16 template <class T, class CMP = std::less<T>, class
  ALLOC = std::allocator<T> >
17 std::set<T, CMP, ALLOC> operator + (
18     const std::set<T, CMP, ALLOC> &s1, const std::set<T
  , CMP, ALLOC> &s2)
19 {
20     std::set<T, CMP, ALLOC> s;
21     std::set_union(s1.begin(), s1.end(), s2.begin(), s2
22     .end(),
23     std::inserter(s, s.begin()));
24     return s;
25 }

```

5 General

5.1 Next Permutation

```

1 // output: 1,2,3; 1,3,2; 2,1,3; 2,3,1; 3,1,2; 3,2,1;
2
3 vector<int> arr = {1, 2, 3};
4 int n = arr.size();
5
6 do {
7     for (auto e : arr) {
8         cout << e << ' ';
9     }
10    cout << '\n';
11 } while (next_permutation(arr.begin(), arr.end()));

```

5.2 Xor Basis

```

1 // XOR Basis
2 // You are given a set of $N$ integer values. You
  should find the minimum number of values that you
  need to add to the set such that the following
  will hold true:
3 // For every two integers $A$ and $B$ in the set,
  their bitwise xor $A \oplus B$ is also in the set
4
5 vector<ll> basis;
6
7 void add(ll x) {
8     for (int i = 0; i < (int)basis.size(); i++) {
9         // reduce x using the current basis vectors
10        x = min(x, x ^ basis[i]);
11    }
12
13    if (x != 0) { basis.push_back(x); }
14 }
15
16 ll res = (1LL << (int)basis.size()) - n;

```

5.3 Min Priority Queue

```
1 template<class T> using min_priority_queue =
  priority_queue<T, vector<T>, greater<T>>;
```

5.4 Custom Unordered Map

```
1 // Source: Tiagosf00
2
3 struct custom_hash {
4     static uint64_t splitmix64(uint64_t x) {
5         // http://xorshift.di.unimi.it/splitmix64.c
6         x += 0x9e3779b97f4a7c15;
7         x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
8         x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
9         return x ^ (x >> 31);
10    }
11
12    size_t operator()(uint64_t x) const {
13        static const uint64_t FIXED_RANDOM = chrono::
14        steady_clock::now().time_since_epoch().count();
15        return splitmix64(x + FIXED_RANDOM);
16    }
17 };
18 unordered_map<long long, int, custom_hash> safe_map;
19
20 // when using pairs
21 struct custom_hash {
22     inline size_t operator()(const pii & a) const {
23         return (a.first << 6) ^ (a.first >> 2) ^
24         2038074743 ^ a.second;
25     }
26 };
```

5.5 Flags

```
1 // g++ -std=c++17 -Wall -Wshadow -fsanitize=address -
  02 -D -o cod a.cpp
```

5.6 First True

```
1 // Binary Search (first_true)
2 //
3 // first_true(2, 10, [](int x) { return x * x >= 30;
4 // }); // outputs 6
5 //
6 // [1, r]
7 //
8 // if none of the values in the range work, return hi
9 // + 1
10 //
11 // f(4) = false
12 // f(5) = false
13 // f(6) = true
14 // f(7) = true
15
16 int first_true(int lo, int hi, function<bool(int)> f)
17 {
18     hi++;
19     while (lo < hi) {
20         int mid = lo + (hi - lo) / 2;
21
22         if (f(mid)) {
23             hi = mid;
24         } else {
25             lo = mid + 1;
26         }
27     }
28     return lo;
29 }
```

5.7 Kosaraju

```
1 struct Kosaraju {
2
3     int N;
4     int cntComps;
5
6     vector<vector<int>>> g;
7     vector<vector<int>>> gi;
8
9     stack<int> S;
10    vector<int> vis;
11    vector<int> comp;
12
13    Kosaraju(vector<vector<int>>& arr) {
14        N = (int)arr.size();
15        cntComps = 0;
16
17        g.resize(N);
18        gi.resize(N);
19        vis.resize(N);
20        comp.resize(N);
21
22        for(int i = 0; i < (int)arr.size(); i++) {
23            for(auto &v : arr[i]) {
24                g[i].push_back(v);
25                gi[v].push_back(i);
26            }
27        }
28
29        run();
30    }
31
32    void dfs(int u) {
33        vis[u] = 1;
34        for(auto &v : g[u]) if(!vis[v]) {
35            dfs(v);
36        }
37        S.push(u);
38    }
39
40    void scc(int u, int c) {
41        vis[u] = 1;
42        comp[u] = c;
43        for(auto &v : gi[u]) if(!vis[v]) {
44            scc(v, c);
45        }
46    }
47
48    void run() {
49        vis.assign(N, 0);
50
51        for(int i = 0; i < N; i++) if(!vis[i]) {
52            dfs(i);
53        }
54
55        vis.assign(N, 0);
56
57        while((int)S.size()) {
58            int u = S.top();
59            S.pop();
60            if(!vis[u]) {
61                scc(u, cntComps++);
62            }
63        }
64    }
65 }
66
67 };
```

5.8 Base Converter

```

1  const string digits = "0123456789
   ABCDEFGHIJKLMNOPQRSTUVWXYZ";
2
3  ll tobase10(string number, int base) {
4      map<char, int> val;
5      for (int i = 0; i < digits.size(); i++) {
6          val[digits[i]] = i;
7      }
8
9      ll ans = 0, pot = 1;
10
11     for (int i = number.size() - 1; i >= 0; i--) {
12         ans += val[number[i]] * pot;
13         pot *= base;
14     }
15
16     return ans;
17 }
18
19 string frombase10(ll number, int base) {
20     if (number == 0) return "0";
21
22     string ans = "";
23
24     while (number > 0) {
25         ans += digits[number % base];
26         number /= base;
27     }
28
29     reverse(ans.begin(), ans.end());
30
31     return ans;
32 }
33
34 // verifica se um número está na base especificada
35 bool verify_base(string num, int base) {
36     map<char, int> val;
37     for (int i = 0; i < digits.size(); i++) {
38         val[digits[i]] = i;
39     }
40
41     for (auto digit : num) {
42         if (val[digit] >= base) {
43             return false;
44         }
45     }
46
47     return true;
48 }

```

5.9 Overflow

```

1  // Signatures of some built-in functions to perform
   arithmetic operations with overflow check
2  // Source: https://gcc.gnu.org/onlinedocs/gcc/Integer
   -Overflow-Builtins.html
3  //
4  // you can also check overflow by performing the
   operation with double
5  // and checking if the result it's greater than the
   maximum value supported by the variable
6
7  bool __builtin_add_overflow (type1 a, type2 b, type3
   *res)
8  bool __builtin_sadd_overflow (int a, int b, int *res)
9  bool __builtin_saddl_overflow (long int a, long int b
   , long int *res)
10 bool __builtin_saddll_overflow (long long int a, long
   long int b, long long int *res)
11 bool __builtin_uadd_overflow (unsigned int a,
   unsigned int b, unsigned int *res)
12 bool __builtin_uaddl_overflow (unsigned long int a,
   unsigned long int b, unsigned long int *res)

```

```

13 bool __builtin_uaddll_overflow (unsigned long long
   int a, unsigned long long int b, unsigned long
   long int *res)
14
15 bool __builtin_sub_overflow (type1 a, type2 b, type3
   *res)
16 bool __builtin_ssub_overflow (int a, int b, int *res)
17 bool __builtin_ssubl_overflow (long int a, long int b
   , long int *res)
18 bool __builtin_ssubll_overflow (long long int a, long
   long int b, long long int *res)
19 bool __builtin_usub_overflow (unsigned int a,
   unsigned int b, unsigned int *res)
20 bool __builtin_usubl_overflow (unsigned long int a,
   unsigned long int b, unsigned long int *res)
21 bool __builtin_usubll_overflow (unsigned long long
   int a, unsigned long long int b, unsigned long
   long int *res)
22
23 bool __builtin_mul_overflow (type1 a, type2 b, type3
   *res)
24 bool __builtin_smul_overflow (int a, int b, int *res)
25 bool __builtin_smull_overflow (long int a, long int b
   , long int *res)
26 bool __builtin_smulll_overflow (long long int a, long
   long int b, long long int *res)
27 bool __builtin_umul_overflow (unsigned int a,
   unsigned int b, unsigned int *res)
28 bool __builtin_umull_overflow (unsigned long int a,
   unsigned long int b, unsigned long int *res)
29 bool __builtin_umulll_overflow (unsigned long long
   int a, unsigned long long int b, unsigned long
   long int *res)
30
31 bool __builtin_add_overflow_p (type1 a, type2 b,
   type3 c)
32 bool __builtin_sub_overflow_p (type1 a, type2 b,
   type3 c)
33 bool __builtin_mul_overflow_p (type1 a, type2 b,
   type3 c)

```

5.10 Random

```

1  int main() {
2      ios::sync_with_stdio(false);
3      cin.tie(NULL);
4
5      //mt1937 rng(chrono::steady_clock::now().
   time_since_epoch().count()); //gerar int
6      mt1937_64 rng(chrono::steady_clock::now().
   time_since_epoch().count()); //gerar ll
7
8      /*usar rng() pra gerar numeros aleatórios*/
9      /*usar rng() % x pra gerar numeros em [0, x-1]*/
10     for(int i = 0; i < 10; i++){
11         cout << rng() << endl;
12     }
13     vector<ll> arr = {1,2,3,4,5,6,7,8,9};
14     /*dã pra usar no shuffle de vector também*/
15     shuffle(arr.begin(), arr.end(),rng);
16     for(auto &x: arr)
17         cout << x << endl;
18 }

```

5.11 Get Subsets Sum Iterative

```

1  vector<ll> get_subset_sums(int l, int r, vector<ll>&
   arr) {
2      vector<ll> ans;
3
4      int len = r-l+1;

```

```

5     for (int i = 0; i < (1 << len); i++) {
6         ll sum = 0;
7
8         for (int j = 0; j < len; j++) {
9             if (i & (1 << j)) {
10                sum += arr[1 + j];
11            }
12        }
13
14        ans.push_back(sum);
15    }
16
17    return ans;
18 }

```

5.12 Interactive

```

1 // you should use cout.flush() every cout
2 int query(int a) {
3     cout << "? " << a << '\n';
4     cout.flush();
5     char res; cin >> res;
6     return res;
7 }
8
9 // using endl you don't need
10 int query(int a) {
11     cout << "? " << a << endl;
12     char res; cin >> res;
13     return res;
14 }

```

5.13 Xor 1 To N

```

1 // XOR sum from 1 to N
2 ll xor_1_to_n(ll n) {
3     if (n % 4 == 0) {
4         return n;
5     } else if (n % 4 == 1) {
6         return 1;
7     } else if (n % 4 == 2) {
8         return n + 1;
9     }
10
11     return 0;
12 }

```

5.14 Input By File

```

1 freopen("file.in", "r", stdin);
2 freopen("file.out", "w", stdout);

```

5.15 Template

```

1 #include <bits/stdc++.h>
2 #define ff first
3 #define ss second
4
5 using namespace std;
6 using ll = long long;
7 using ld = long double;
8 using pii = pair<int,int>;
9 using vi = vector<int>;
10
11 using tii = tuple<int,int,int>;
12 // auto [a,b,c] = ...
13 // .insert({a,b,c})
14
15 const int oo = (int)1e9 + 5; //INF to INT
16 const ll OO = 0x3f3f3f3f3f3f3fLL; //INF to LL
17

```

```

18 // g++ -std=c++17 -Wall -Wshadow -fsanitize = address
19 // -O2 -o cod a.cpp
20
21 int main() {
22     ios::sync_with_stdio(false);
23     cin.tie(NULL);
24
25
26     return 0;
27 }

```

5.16 Mix Hash

```

1 // magic hash function using mix
2
3 using ull = unsigned long long;
4 ull mix(ull o){
5     o+=0x9e3779b97f4a7c15;
6     o=(o^(o>>30))*0xbf58476d1ce4e5b9;
7     o=(o^(o>>27))*0x94d049bb133111eb;
8     return o^(o>>31);
9 }
10 ull hash(pii a) {return mix(a.first ^ mix(a.second))
11 ;}

```

5.17 Last True

```

1 // Binary Search (last_true)
2
3 // last_true(2, 10, [](int x) { return x * x <= 30;
4 // }); // outputs 5
5
6 // [l, r]
7 // if none of the values in the range work, return lo
8 // - 1
9
10 // f(1) = true
11 // f(2) = true
12 // f(3) = true
13 // f(4) = true
14 // f(5) = true
15 // f(6) = false
16 // f(7) = false
17 // f(8) = false
18
19 // last_true(1, 8, f) = 5
20 // last_true(7, 8, f) = 6
21
22 int last_true(int lo, int hi, function<bool(int)> f)
23 {
24     lo--;
25     while (lo < hi) {
26         int mid = lo + (hi - lo + 1) / 2;
27
28         if (f(mid)) {
29             lo = mid;
30         } else {
31             hi = mid - 1;
32         }
33     }
34     return lo;
35 }

```

6 Geometry

6.1 Convex Hull

```

1 // Convex Hull - Monotone Chain
2 //

```

```

3 // Convex Hull is the subset of points that forms the
  // smallest convex polygon
4 // which encloses all points in the set.
5 //
6 // https://cses.fi/problemset/task/2195/
7 // https://open.kattis.com/problems/convexhull (
  counterclockwise)
8 //
9 // O(n log(n))
10
11 typedef long long ftype;
12
13 struct Point {
14     ftype x, y;
15
16     Point() {} ;
17     Point(ftype x, ftype y) : x(x), y(y) {} ;
18
19     bool operator<(Point o) {
20         if (x == o.x) return y < o.y;
21         return x < o.x;
22     }
23
24     bool operator==(Point o) {
25         return x == o.x && y == o.y;
26     }
27 };
28
29 ftype cross(Point a, Point b, Point c) {
30     // v: a -> c
31     // w: a -> b
32
33     // v: c.x - a.x, c.y - a.y
34     // w: b.x - a.x, b.y - a.y
35
36     return (c.x - a.x) * (b.y - a.y) - (c.y - a.y) *
37         (b.x - a.x);
38 }
39
40 ftype dir(Point a, Point b, Point c) {
41     // 0 -> colineares
42     // -1 -> esquerda
43     // 1 -> direita
44
45     ftype cp = cross(a, b, c);
46
47     if (cp == 0) return 0;
48     else if (cp < 0) return -1;
49     else return 1;
50 }
51
52 vector<Point> convex_hull(vector<Point> points) {
53     sort(points.begin(), points.end());
54     points.erase( unique(points.begin(), points.end())
55         , points.end()); // somente pontos distintos
56     int n = points.size();
57
58     if (n == 1) return { points[0] };
59
60     vector<Point> upper_hull = {points[0], points
61         [1]};
62     for (int i = 2; i < n; i++) {
63         upper_hull.push_back(points[i]);
64
65         int sz = upper_hull.size();
66
67         while (sz >= 3 && dir(upper_hull[sz-3],
68             upper_hull[sz-2], upper_hull[sz-1]) == -1) {
69             upper_hull.pop_back();
70             upper_hull.pop_back();
71             upper_hull.push_back(points[i]);
72             sz--;
73         }
74     }
75 }

```

```

76     }
77
78     vector<Point> lower_hull = {points[n-1], points[n
79         -2]};
80     for (int i = n-3; i >= 0; i--) {
81         lower_hull.push_back(points[i]);
82
83         int sz = lower_hull.size();
84
85         while (sz >= 3 && dir(lower_hull[sz-3],
86             lower_hull[sz-2], lower_hull[sz-1]) == -1) {
87             lower_hull.pop_back();
88             lower_hull.pop_back();
89             lower_hull.push_back(points[i]);
90             sz--;
91         }
92     }
93
94     // reverse(lower_hull.begin(), lower_hull.end());
95     // counterclockwise
96
97     for (int i = (int)lower_hull.size() - 2; i > 0; i
98         --) {
99         upper_hull.push_back(lower_hull[i]);
100     }
101
102     return upper_hull;
103 }

```

7 Graph

7.1 Floyd Warshall

```

1 const long long LLINF = 0x3f3f3f3f3f3f3f3fLL;
2
3 for (int i = 0; i < n; i++) {
4     for (int j = 0; j < n; j++) {
5         adj[i][j] = 0;
6     }
7 }
8
9 long long dist[MAX][MAX];
10 for (int i = 0; i < n; i++) {
11     for (int j = 0; j < n; j++) {
12         if (i == j)
13             dist[i][j] = 0;
14         else if (adj[i][j])
15             dist[i][j] = adj[i][j];
16         else
17             dist[i][j] = LLINF;
18     }
19 }
20
21 for (int k = 0; k < n; k++) {
22     for (int i = 0; i < n; i++) {
23         for (int j = 0; j < n; j++) {
24             dist[i][j] = min(dist[i][j], dist[i][k] +
25                 dist[k][j]);
26         }
27     }
28 }

```

7.2 Ford Fulkerson

```

1 // Ford-Fulkerson
2 //
3 // max-flow / min-cut
4 //
5 // MAX nÃss
6 //
7 // https://cses.fi/problemset/task/1694/

```



```

8 //
9 // O(m * max_flow)
10
11 using ll = long long;
12 const int MAX = 510;
13
14 struct Flow {
15     int n;
16     ll adj[MAX][MAX];
17     bool used[MAX];
18
19     Flow(int n) : n(n) {};
20
21     void add_edge(int u, int v, ll c) {
22         adj[u][v] += c;
23         adj[v][u] = 0; // cuidado com isso
24     }
25
26     ll dfs(int x, int t, ll amount) {
27         used[x] = true;
28
29         if (x == t) return amount;
30
31         for (int i = 1; i <= n; i++) {
32             if (adj[x][i] > 0 && !used[i]) {
33                 ll sent = dfs(i, t, min(amount, adj[x
34 ][i]));
35
36                 if (sent > 0) {
37                     adj[x][i] -= sent;
38                     adj[i][x] += sent;
39
40                     return sent;
41                 }
42             }
43         }
44         return 0;
45     }
46
47     ll max_flow(int s, int t) { // source and sink
48         ll total = 0;
49         ll sent = -1;
50
51         while (sent != 0) {
52             memset(used, 0, sizeof(used));
53             sent = dfs(s, t, INT_MAX);
54             total += sent;
55         }
56
57         return total;
58     }
59 };

```

7.3 Dinic

```

1 // Dinic / Dinitz
2 //
3 // max-flow / min-cut
4 //
5 // https://cses.fi/problemset/task/1694/
6 //
7 // O(E * V^2)
8
9 using ll = long long;
10 const ll FLOW_INF = 1e18 + 7;
11
12 struct Edge {
13     int from, to;
14     ll cap, flow;
15     Edge* residual; // a inversa da minha aresta
16
17     Edge() {};

```

```

18
19     Edge(int from, int to, ll cap) : from(from), to(
20 to), cap(cap), flow(0) {};
21
22     ll remaining_cap() {
23         return cap - flow;
24     }
25
26     void augment(ll bottle_neck) {
27         flow += bottle_neck;
28         residual->flow -= bottle_neck;
29     }
30
31     bool is_residual() {
32         return cap == 0;
33     }
34 };
35
36 struct Dinic {
37     int n;
38     vector<vector<Edge*>> adj;
39     vector<int> level, next;
40
41     Dinic(int n): n(n) {
42         adj.assign(n+1, vector<Edge*>());
43         level.assign(n+1, -1);
44         next.assign(n+1, 0);
45     }
46
47     void add_edge(int from, int to, ll cap) {
48         auto e1 = new Edge(from, to, cap);
49         auto e2 = new Edge(to, from, 0);
50
51         e1->residual = e2;
52         e2->residual = e1;
53
54         adj[from].push_back(e1);
55         adj[to].push_back(e2);
56     }
57
58     bool bfs(int s, int t) {
59         fill(level.begin(), level.end(), -1);
60         queue<int> q;
61
62         q.push(s);
63         level[s] = 1;
64
65         while (q.size()) {
66             int curr = q.front();
67             q.pop();
68
69             for (auto edge : adj[curr]) {
70                 if (edge->remaining_cap() > 0 &&
71 level[edge->to] == -1) {
72                     level[edge->to] = level[curr] +
73 1;
74                     q.push(edge->to);
75                 }
76             }
77         }
78
79         return level[t] != -1;
80     }
81
82     ll dfs(int x, int t, ll flow) {
83         if (x == t) return flow;
84
85         for (int& cid = next[x]; cid < (int)adj[x].
86 size(); cid++) {
87             auto& edge = adj[x][cid];
88             ll cap = edge->remaining_cap();
89
90             if (cap > 0 && level[edge->to] == level[x

```

```

] + 1) {
    ll sent = dfs(edge->to, t, min(flow,
cap)); // bottle neck
    if (sent > 0) {
        edge->augment(sent);
        return sent;
    }
}
}
return 0;
}

ll solve(int s, int t) {
    ll max_flow = 0;

    while (bfs(s, t)) {
        fill(next.begin(), next.end(), 0);

        while (ll sent = dfs(s, t, FLOW_INF)) {
            max_flow += sent;
        }

        return max_flow;
    }

    // path recover
    vector<bool> vis;
    vector<int> curr;

    bool dfs2(int x, int& t) {
        vis[x] = true;
        bool arrived = false;

        if (x == t) {
            curr.push_back(x);
            return true;
        }

        for (auto e : adj[x]) {
            if (e->flow > 0 && !vis[e->to]) { // !e->
is_residual() &&
                bool aux = dfs2(e->to, t);

                if (aux) {
                    arrived = true;
                    e->flow--;
                }
            }

            if (arrived) curr.push_back(x);

            return arrived;
        }
    }

    vector<vector<int>> get_paths(int s, int t) {
        vector<vector<int>> ans;

        while (true) {
            curr.clear();
            vis.assign(n+1, false);

            if (!dfs2(s, t)) break;

            reverse(curr.begin(), curr.end());
            ans.push_back(curr);
        }

        return ans;
    }
};

```

7.4 Lca

```

1 // LCA
2 //
3 // lowest common ancestor between two nodes
4 //
5 // edit_distance(n, adj, root)
6 //
7 // https://cses.fi/problemset/task/1688
8 //
9 // O(log N)
10
11 struct LCA {
12     const int MAXE = 31;
13     vector<vector<int>> up;
14     vector<int> dep;
15
16     LCA(int n, vector<vector<int>>& adj, int root =
1) {
17         up.assign(n+1, vector<int>(MAXE, -1));
18         dep.assign(n+1, 0);
19
20         dep[root] = 1;
21         dfs(root, -1, adj);
22
23         for (int j = 1; j < MAXE; j++) {
24             for (int i = 1; i <= n; i++) {
25                 if (up[i][j-1] != -1)
26                     up[i][j] = up[ up[i][j-1] ][j-1];
27             }
28         }
29     }
30
31     void dfs(int x, int p, vector<vector<int>>& adj)
32     {
33         up[x][0] = p;
34         for (auto e : adj[x]) {
35             if (e != p) {
36                 dep[e] = dep[x] + 1;
37                 dfs(e, x, adj);
38             }
39         }
40     }
41
42     int jump(int x, int k) { // jump from node x k
43         times
44         for (int i = 0; i < MAXE; i++) {
45             if (k && (1 << i) && x != -1) x = up[x][i];
46         }
47         return x;
48     }
49
50     int lca(int a, int b) {
51         if (dep[a] > dep[b]) swap(a, b);
52         b = jump(b, dep[b] - dep[a]);
53
54         if (a == b) return a;
55
56         for (int i = MAXE-1; i >= 0; i--) {
57             if (up[a][i] != up[b][i]) {
58                 a = up[a][i];
59                 b = up[b][i];
60             }
61         }
62         return up[a][0];
63     }
64
65     int dist(int a, int b) {
66         return dep[a] + dep[b] - 2 * dep[lca(a, b)];
67     }
};

```

7.5 3sat

```

1 // We are given a CNF, e.g. phi(x) = (x_1 or ~x_2)
  // and (x_3 or ~x_4 or ~x_5) and ...
2 // SAT finds an assignment x for phi(x) = true.
3 // Davis-Putnum-Logemann-Loveland Algorithm (
  // youknowwho code)
4 // Complexity: O(2^n) in worst case.
5 // This implementation is practical for n <= 1000 or
  // more. lmao.
6
7 #include<bits/stdc++.h>
8 using namespace std;
9
10 const int N = 3e5 + 9;
11
12 // positive literal x in [0,n),
13 // negative literal ~x in [-n,0)
14 // 0 indexed
15 struct SAT_GOD {
16     int n;
17     vector<int> occ, pos, neg;
18     vector<vector<int>>> g, lit;
19     SAT_GOD(int n) : n(n), g(2*n), occ(2*n) {}
20     void add_clause(const vector<int> &c) {
21         for(auto u: c) {
22             g[u+n].push_back(lit.size());
23             occ[u+n] += 1;
24         }
25         lit.push_back(c);
26     }
27     //(!u | v | !w) -> (u, 0, v, 1, w, 0)
28     void add(int u, int af, int v = 1e9, int bf = 0,
29             int w = 1e9, int cf = 0) {
30         vector<int> a;
31         if(!af) u = ~u;
32         a.push_back(u);
33         if(v != 1e9) {
34             if(!bf) v = ~v;
35             a.push_back(v);
36         }
37         if(w != 1e9) {
38             if(!cf) w = ~w;
39             a.push_back(w);
40         }
41         add_clause(a);
42     }
43     vector<bool> x;
44     vector<vector<int>>> decision_stack;
45     vector<int> unit_stack, pure_stack;
46     void push(int u) {
47         x[u + n] = 1;
48         decision_stack.back().push_back(u);
49         for (auto i: g[u + n]) if (pos[i]++ == 0) {
50             for (auto u: lit[i]) --occ[u+n];
51         }
52         for (auto i: g[~u + n]) {
53             ++neg[i];
54             if (pos[i] == 0) unit_stack.push_back(i);
55         }
56     }
57     void pop() {
58         int u = decision_stack.back().back();
59         decision_stack.back().pop_back();
60         x[u + n] = 0;
61         for (auto i: g[u + n]) if (--pos[i] == 0) {
62             for (auto u: lit[i]) ++occ[u + n];
63         }
64         for (auto i: g[~u+n]) --neg[i];
65     }
66     bool reduction() {
67         while(!unit_stack.empty() || !pure_stack.empty()) {

```

```

67         if(!pure_stack.empty()) { // pure literal
68             elimination
69             int u = pure_stack.back();
70             pure_stack.pop_back();
71             if (occ[u + n] == 1 && occ[~u + n] == 0) push
72             (u);
73         } else { // unit propagation
74             int i = unit_stack.back();
75             unit_stack.pop_back();
76             if(pos[i] > 0) continue;
77             if(neg[i] == lit[i].size()) return false;
78             if(neg[i] + 1 == lit[i].size()) {
79                 int w = n;
80                 for (int u: lit[i]) if (!x[u + n] && !x[~u
81                 + n]) w = u;
82                 if (x[~w + n]) return false;
83                 push(w);
84             }
85         }
86         return true;
87     }
88     bool ok() {
89         x.assign(2*n,0);
90         pos = neg = vector<int>(lit.size());
91         decision_stack.assign(1, {});
92         while(1) {
93             if(reduction()) {
94                 int s = 0;
95                 for(int u = 0; u < n; ++u) if(occ[s + n] +
96                 occ[~s + n] < occ[u + n] + occ[~u + n]) s = u;
97                 if(occ[s + n] + occ[~s + n] == 0) return true
98                 ;
99                 decision_stack.push_back({});
100                 push(s);
101             } else {
102                 int s = decision_stack.back()[0];
103                 while(!decision_stack.back().empty()) pop();
104                 decision_stack.pop_back();
105                 if (decision_stack.empty()) return false;
106                 push(~s);
107             }
108         }
109     }
110 }
111
112 int32_t main() {
113     int n = 9;
114     SAT_GOD t(n);
115     t.add(0, 0, 1, 1);
116     t.add(1, 0);
117     t.add(1, 0, 3, 1, 5, 1);
118     cout << t.ok() << endl;
119 }

```

7.6 Dijkstra

```

1 const int INF = 1e9+17;
2 vector<vector<pair<int, int>>> adj; // {neighbor,
  // weight}
3
4 void dijkstra(int s, vector<int> & d, vector<int> & p
5 ) {
6     int n = adj.size();
7     d.assign(n, INF);
8     p.assign(n, -1);
9
10    d[s] = 0;
11    set<pair<int, int>> q;
12    q.insert({0, s});
13    while (!q.empty()) {
14        int v = q.begin()->second;
15        q.erase(q.begin());

```

```

15         for (auto edge : adj[v]) {
16             int to = edge.first;
17             int len = edge.second;
18
19             if (d[v] + len < d[to]) {
20                 q.erase({d[to], to});
21                 d[to] = d[v] + len;
22                 p[to] = v;
23                 q.insert({d[to], to});
24             }
25         }
26     }
27 }
28 }

```

7.7 Has Negative Cycle

```

1 // Edson
2
3 using edge = tuple<int, int, int>;
4
5 bool has_negative_cycle(int s, int N, const vector<
6     edge>& edges)
7 {
8     const int INF { 1e9+17 };
9     vector<int> dist(N + 1, INF);
10    dist[s] = 0;
11
12    for (int i = 1; i <= N - 1; i++) {
13        for (auto [u, v, w] : edges) {
14            if (dist[u] < INF && dist[v] > dist[u] +
15                w) {
16                dist[v] = dist[u] + w;
17            }
18        }
19    }
20
21    for (auto [u, v, w] : edges) {
22        if (dist[u] < INF && dist[v] > dist[u] + w) {
23            return true;
24        }
25    }
26    return false;
27 }

```

7.8 2sat

```

1 // 2SAT
2 //
3 // verifica se existe e encontra solu~ao
4 // para f~ormulas booleanas da forma
5 // (a or b) and (!a or c) and (...)
6 //
7 // indexado em 0
8 // n(a) = 2*x e n(~a) = 2*x+1
9 // a = 2 ; n(a) = 4 ; n(~a) = 5 ; n(a)^1 = 5 ; n(~a)
10 // ^1 = 4
11 //
12 // https://cses.fi/problemset/task/1684/
13 // https://codeforces.com/gym/104120/problem/E
14 // (add_eq, add_true, add_false e at_most_one n~ao
15 // foram testadas)
16 //
17 // 0(n + m)
18
19 struct sat {
20     int n, tot;
21     vector<vector<int>> adj, adjt; // grafo original,
22     grafo transposto
23     vector<int> vis, comp, ans;

```

```

21     stack<int> topo; // ordem topol~ogica
22
23     sat() {}
24     sat(int n_) : n(n_), tot(n), adj(2*n), adjt(2*n)
25     {}
26
27     void dfs(int x) {
28         vis[x] = true;
29
30         for (auto e : adj[x]) {
31             if (!vis[e]) dfs(e);
32         }
33
34         topo.push(x);
35     }
36
37     void dfst(int x, int& id) {
38         vis[x] = true;
39         comp[x] = id;
40
41         for (auto e : adjt[x]) {
42             if (!vis[e]) dfst(e, id);
43         }
44     }
45
46     void add_impl(int a, int b) { // a -> b = (!a or
47         b)
48         a = (a >= 0 ? 2*a : -2*a-1);
49         b = (b >= 0 ? 2*b : -2*b-1);
50
51         adj[a].push_back(b);
52         adj[b^1].push_back(a^1);
53
54         adjt[b].push_back(a);
55         adjt[a^1].push_back(b^1);
56     }
57
58     void add_or(int a, int b) { // a or b
59         add_impl(~a, b);
60     }
61
62     void add_nor(int a, int b) { // a nor b = !(a or
63         b)
64         add_or(~a, b), add_or(a, ~b), add_or(~a, ~b);
65     }
66
67     void add_and(int a, int b) { // a and b
68         add_or(a, b), add_or(~a, b), add_or(a, ~b);
69     }
70
71     void add_nand(int a, int b) { // a nand b = !(a
72         and b)
73         add_or(~a, ~b);
74     }
75
76     void add_xor(int a, int b) { // a xor b = (a != b)
77         add_or(a, b), add_or(~a, ~b);
78     }
79
80     void add_xnor(int a, int b) { // a xnor b = !(a
81         xor b) = (a = b)
82         add_xor(~a, b);
83     }
84
85     void add_true(int a) { // a = T
86         add_or(a, ~a);
87     }
88
89     void add_false(int a) { // a = F
90         add_and(a, ~a);
91     }

```

```

88 // magia - brunomaletta
89 void add_true_old(int a) { // a = T (n sei se
funciona)
90     add_impl(~a, a);
91 }
92
93 void at_most_one(vector<int> v) { // no max um
verdadeiro
94     adj.resize(2*(tot+v.size()));
95     for (int i = 0; i < v.size(); i++) {
96         add_impl(tot+i, ~v[i]);
97         if (i) {
98             add_impl(tot+i, tot+i-1);
99             add_impl(v[i], tot+i-1);
100         }
101     }
102     tot += v.size();
103 }
104
105 pair<bool, vector<int>> solve() {
106     ans.assign(n, -1);
107     comp.assign(2*tot, -1);
108     vis.assign(2*tot, 0);
109     int id = 1;
110
111     for (int i = 0; i < 2*tot; i++) if (!vis[i])
dfs(i);
112
113     vis.assign(2*tot, 0);
114     while (topo.size()) {
115         auto x = topo.top();
116         topo.pop();
117
118         if (!vis[x]) {
119             dfst(x, id);
120             id++;
121         }
122     }
123
124     for (int i = 0; i < tot; i++) {
125         if (comp[2*i] == comp[2*i+1]) return {
false, {} };
126         ans[i] = (comp[2*i] > comp[2*i+1]);
127     }
128
129     return {true, ans};
130 }
131 };

```

7.9 Min Cost Max Flow

```

1 // Min Cost Max Flow (brunomaletta)
2 //
3 // min_cost_flow(s, t, f) computa o par (fluxo, custo
)
4 // com max(fluxo) <= f que tenha min(custo)
5 // min_cost_flow(s, t) -> Fluxo maximo de custo
minimo de s pra t
6 // Se for um dag, da pra substituir o SPFA por uma DP
pra nao
7 // pagar O(nm) no comeco
8 // Se nao tiver aresta com custo negativo, nao
precisa do SPFA
9 //
10 // O(nm + f * m log n)
11
12 template<typename T> struct mcmf {
13     struct edge {
14         int to, rev, flow, cap; // para, id da
reversa, fluxo, capacidade
15         bool res; // se eh reversa
16         T cost; // custo da unidade de fluxo

```

```

17         edge() : to(0), rev(0), flow(0), cap(0), cost
(0), res(false) {}
18         edge(int to_, int rev_, int flow_, int cap_,
T cost_, bool res_)
19             : to(to_), rev(rev_), flow(flow_), cap(
cap_), res(res_), cost(cost_) {}
20     };
21
22     vector<vector<edge>> g;
23     vector<int> par_idx, par;
24     T inf;
25     vector<T> dist;
26
27     mcmf(int n) : g(n), par_idx(n), par(n), inf(
numeric_limits<T>::max()/3) {}
28
29     void add(int u, int v, int w, T cost) { // de u
pra v com cap w e custo cost
30         edge a = edge(v, g[v].size(), 0, w, cost,
false);
31         edge b = edge(u, g[u].size(), 0, 0, -cost,
true);
32
33         g[u].push_back(a);
34         g[v].push_back(b);
35     }
36
37     vector<T> spfa(int s) { // nao precisa se nao
tiver custo negativo
38         deque<int> q;
39         vector<bool> is_inside(g.size(), 0);
40         dist = vector<T>(g.size(), inf);
41
42         dist[s] = 0;
43         q.push_back(s);
44         is_inside[s] = true;
45
46         while (!q.empty()) {
47             int v = q.front();
48             q.pop_front();
49             is_inside[v] = false;
50
51             for (int i = 0; i < g[v].size(); i++) {
52                 auto [to, rev, flow, cap, res, cost]
= g[v][i];
53                 if (flow < cap and dist[v] + cost <
dist[to]) {
54                     dist[to] = dist[v] + cost;
55
56                     if (is_inside[to]) continue;
57                     if (!q.empty() and dist[to] >
dist[q.front()]) q.push_back(to);
58                     else q.push_front(to);
59                     is_inside[to] = true;
60                 }
61             }
62             return dist;
63         }
64
65     bool dijkstra(int s, int t, vector<T>& pot) {
66         priority_queue<pair<T, int>, vector<pair<T,
int>>, greater<>> q;
67         dist = vector<T>(g.size(), inf);
68         dist[s] = 0;
69         q.emplace(0, s);
70         while (q.size()) {
71             auto [d, v] = q.top();
72             q.pop();
73             if (dist[v] < d) continue;
74             for (int i = 0; i < g[v].size(); i++) {
75                 auto [to, rev, flow, cap, res, cost]
= g[v][i];
76                 cost += pot[v] - pot[to];

```

```

77         if (flow < cap and dist[v] + cost <
dist[to]) {
78             dist[to] = dist[v] + cost;
79             q.emplace(dist[to], to);
80             par_idx[to] = i, par[to] = v;
81         }
82     }
83 }
84 return dist[t] < inf;
85 }
86
87 pair<int, T> min_cost_flow(int s, int t, int flow
= INF) {
88     vector<T> pot(g.size(), 0);
89     pot = spfa(s); // mudar algoritmo de caminho
minimo aqui
90
91     int f = 0;
92     T ret = 0;
93     while (f < flow and dijkstra(s, t, pot)) {
94         for (int i = 0; i < g.size(); i++)
95             if (dist[i] < inf) pot[i] += dist[i];
96
97         int mn_flow = flow - f, u = t;
98         while (u != s) {
99             mn_flow = min(mn_flow,
100 g[par[u]][par_idx[u]].cap - g[par
[u]][par_idx[u]].flow);
101             u = par[u];
102         }
103
104         ret += pot[t] * mn_flow;
105
106         u = t;
107         while (u != s) {
108             g[par[u]][par_idx[u]].flow += mn_flow
;
109             g[u][g[par[u]][par_idx[u]].rev].flow
-= mn_flow;
110             u = par[u];
111         }
112
113         f += mn_flow;
114     }
115
116     return make_pair(f, ret);
117 }
118
119 // Opcional: retorna as arestas originais por
onde passa flow = cap
120 vector<pair<int, int>> recover() {
121     vector<pair<int, int>> used;
122     for (int i = 0; i < g.size(); i++) for (edge
e : g[i])
123         if (e.flow == e.cap && !e.res) used.
push_back({i, e.to});
124     return used;
125 }
126 };

```

7.10 Bfs

```

1 vector<vector<int>> adj; // adjacency list
representation
2 int n; // number of nodes
3 int s; // source vertex
4
5 queue<int> q;
6 vector<bool> used(n + 1);
7 vector<int> d(n + 1), p(n + 1);
8
9 q.push(s);
10 used[s] = true;

```

```

11 p[s] = -1;
12 while (!q.empty()) {
13     int v = q.front();
14     q.pop();
15     for (int u : adj[v]) {
16         if (!used[u]) {
17             used[u] = true;
18             q.push(u);
19             d[u] = d[v] + 1;
20             p[u] = v;
21         }
22     }
23 }
24
25 // restore path
26 if (!used[u]) {
27     cout << "No path!";
28 } else {
29     vector<int> path;
30
31     for (int v = u; v != -1; v = p[v])
32         path.push_back(v);
33
34     reverse(path.begin(), path.end());
35
36     cout << "Path: ";
37     for (int v : path)
38         cout << v << " ";
39 }

```

8 DP

8.1 Lcs

```

1 // LCS (Longest Common Subsequence)
2 //
3 // maior subsequencia comum entre duas strings
4 //
5 // tamanho da matriz da dp eh |a| x |b|
6 // lcs(a, b) = string da melhor resposta
7 // dp[a.size()][b.size()] = tamanho da melhor
resposta
8 //
9 // https://atcoder.jp/contests/dp/tasks/dp_f
10 //
11 // O(n^2)
12
13 string lcs(string a, string b) {
14     int n = a.size();
15     int m = b.size();
16
17     int dp[n+1][m+1];
18     pair<int, int> p[n+1][m+1];
19
20     memset(dp, 0, sizeof(dp));
21     memset(p, -1, sizeof(p));
22
23     for (int i = 1; i <= n; i++) {
24         for (int j = 1; j <= m; j++) {
25             if (a[i-1] == b[j-1]) {
26                 dp[i][j] = dp[i-1][j-1] + 1;
27                 p[i][j] = {i-1, j-1};
28             } else {
29                 if (dp[i-1][j] > dp[i][j-1]) {
30                     dp[i][j] = dp[i-1][j];
31                     p[i][j] = {i-1, j};
32                 } else {
33                     dp[i][j] = dp[i][j-1];
34                     p[i][j] = {i, j-1};
35                 }
36             }
37         }
38     }

```

```

38     }
39
40     // recuperar resposta
41
42     string ans = "";
43     pair<int, int> curr = {n, m};
44
45     while (curr.first != 0 && curr.second != 0) {
46         auto [i, j] = curr;
47
48         if (a[i-1] == b[j-1]) {
49             ans += a[i-1];
50         }
51
52         curr = p[i][j];
53     }
54
55     reverse(ans.begin(), ans.end());
56
57     return ans;
58 }

```

8.2 Knapsack

```

1 //Submeter em c++ 64bits otimiza o long long
2 ll knapsack(vector<ll>& weight, vector<ll>& value,
3     int W) {
4     //Usar essa knapsack se sã§ precisar do resultado
5     //final.
6     //O(W) em memã§ria
7     vector<vector<ll>> table(2, vector<ll>(W + 1, 0));
8
9     int n = (int)value.size();
10
11     for(int k = 1; k <= n; k++) {
12         for(int i = 0; i <= W; i++) {
13             if(i - weight[k - 1] >= 0) {
14                 table[k % 2][i] = max(table[(k - 1)
15 % 2][i],
16                 value[k - 1] + table[(k - 1) %
17 2][i - weight[k - 1]]);
18             } else {
19                 table[k % 2][i] = max(table[(k - 1) %
20 2][i], table[k % 2][i]);
21             }
22         }
23     }
24
25     return table[n % 2][W];
26 }
27
28 ll knapsack(vector<ll>& weight, vector<ll>& value,
29     int W) {
30     //Usar essa knapsack se, em algum momento,
31     //precisar recuperar os indices
32     //O(NW) em memã§ria
33
34     int n = (int)value.size();
35     vector<vector<ll>> table(W + 1, vector<ll>(n + 1,
36 0));
37
38     for(int k = 1; k <= n; k++) {
39         for(int i = 0; i <= W; i++) {
40             if(i - weight[k - 1] >= 0) {
41                 table[i][k] = max(table[i][k - 1],
42                 value[k - 1] + table[i - weight[k -
43 1]][k - 1]);
44             } else {
45                 table[i][k] = max(table[i][k - 1],
46                 table[i][k]);
47             }
48         }
49     }
50 }

```

```

39
40 /*
41 int per = W;
42 vector<int> idx;
43 for(int k = n; k > 0; k--) {
44     if(table[per][k] == table[per][k - 1]){
45         continue;
46     } else {
47         idx.push_back(k - 1);
48         per -= weight[k - 1];
49     }
50 }
51 */
52
53 return table[W][n];
54 }
55
56
57 const int MOD = 998244353;
58
59 struct Knapsack {
60
61     int S; // max value
62     vector<ll> dp;
63
64     Knapsack(int S_) {
65         S = S_ + 5;
66         dp.assign(S, 0);
67         dp[0] = 1;
68     }
69
70     void Add(int val) {
71         if(val <= 0 || val >= S) return;
72         for(int i = S - 1; i >= val; i--) {
73             dp[i] += dp[i - val];
74             dp[i] %= MOD;
75         }
76     }
77
78     void Rem(int val) {
79         if(val <= 0 || val >= S) return;
80         for(int i = val; i < S; i++) {
81             dp[i] += MOD - dp[i - val];
82             dp[i] %= MOD;
83         }
84     }
85
86     int Query(int val) {
87         // # of ways to select a subset of numbers
88         with sum = val
89         if(val <= 0 || val >= S) return 0;
90         return dp[val];
91     }
92 };
93
94
95 void solve() {
96
97     int n, w;
98     cin >> n >> w;
99     vector<ll> weight(n), value(n);
100     for(int i = 0; i < n; i++) {
101         cin >> weight[i] >> value[i];
102     }
103     cout << knapsack(weight, value, w) << "\n";
104 }

```

8.3 Lis Binary Search

```

1 int lis(vector<int> arr) {
2     vector<int> dp;
3

```

```

4     for (auto e : arr) {
5         int pos = lower_bound(dp.begin(), dp.end(), e
6         ) - dp.begin();
7
8         if (pos == (int)dp.size()) {
9             dp.push_back(e);
10        } else {
11            dp[pos] = e;
12        }
13    }
14
15    return (int)dp.size();
16 }

```

8.4 Digit Dp 2

```

1 // Digit DP 2: https://cses.fi/problemset/task/2220
2 //
3 // Number of integers between a and b
4 // where no two adjacent digits are the same
5
6 #include <bits/stdc++.h>
7
8 using namespace std;
9 using ll = long long;
10
11 const int MAX = 20; // 10^18
12
13 ll tb[MAX][MAX][2][2];
14
15 ll dp(string& number, int pos, int last_digit, bool
16 under, bool started) {
17     if (pos >= (int)number.size()) {
18         return 1;
19     }
20
21     ll& mem = tb[pos][last_digit][under][started];
22     if (mem != -1) return mem;
23     mem = 0;
24
25     int limit = 9;
26     if (!under) limit = number[pos] - '0';
27
28     for (int digit = 0; digit <= limit; digit++) {
29         if (started && digit == last_digit) continue;
30
31         bool is_under = under || (digit < limit);
32         bool is_started = started || (digit != 0);
33
34         mem += dp(number, pos+1, digit, is_under,
35 is_started);
36     }
37     return mem;
38 }
39
40 ll solve(ll ubound) {
41     memset(tb, -1, sizeof(tb));
42     string number = to_string(ubound);
43     return dp(number, 0, 10, 0, 0);
44 }
45
46 int main() {
47     ios::sync_with_stdio(false);
48     cin.tie(NULL);
49
50     ll a, b; cin >> a >> b;
51     cout << solve(b) - solve(a-1) << '\n';
52
53     return 0;
54 }

```

8.5 Lis Segtree

```

1 int n, arr[MAX], aux[MAX]; cin >> n;
2 for (int i = 0; i < n; i++) {
3     cin >> arr[i];
4     aux[i] = arr[i];
5 }
6
7 sort(aux, aux+n);
8
9 Segtree st(n); // seg of maximum
10
11 int ans = 0;
12 for (int i = 0; i < n; i++) {
13     int it = lower_bound(aux, aux+n, arr[i]) - aux;
14     int lis = st.query(0, it) + 1;
15
16     st.update(it, lis);
17
18     ans = max(ans, lis);
19 }
20
21 cout << ans << '\n';

```

8.6 Edit Distance

```

1 // Edit Distance / Levenshtein Distance
2 //
3 // numero minimo de operacoes
4 // para transformar
5 // uma string em outra
6 //
7 // tamanho da matriz da dp eh |a| x |b|
8 // edit_distance(a.size(), b.size(), a, b)
9 //
10 // https://cses.fi/problemset/task/1639
11 //
12 // O(n^2)
13
14 int tb[MAX][MAX];
15
16 int edit_distance(int i, int j, string &a, string &b)
17 {
18     if (i == 0) return j;
19     if (j == 0) return i;
20
21     int &ans = tb[i][j];
22
23     if (ans != -1) return ans;
24
25     ans = min({
26         edit_distance(i-1, j, a, b) + 1,
27         edit_distance(i, j-1, a, b) + 1,
28         edit_distance(i-1, j-1, a, b) + (a[i-1] != b[
29 j-1])
30     });
31
32     return ans;
33 }

```

8.7 Range Dp

```

1 // Range DP 1: https://codeforces.com/problemset/
2 // problem/1132/F
3 //
4 // You may apply some operations to this string
5 // in one operation you can delete some contiguous
6 // substring of this string
7 // if all letters in the substring you delete are
8 // equal
9 // calculate the minimum number of operations to
10 // delete the whole string s

```



```

7
8 #include <bits/stdc++.h>
9
10 using namespace std;
11
12 const int MAX = 510;
13
14 int n, tb[MAX][MAX];
15 string s;
16
17 int dp(int left, int right) {
18     if (left > right) return 0;
19
20     int& mem = tb[left][right];
21     if (mem != -1) return mem;
22
23     mem = 1 + dp(left+1, right); // gastar uma
24     // operação arrumando só o cara atual
25     for (int i = left+1; i <= right; i++) {
26         if (s[left] == s[i]) {
27             mem = min(mem, dp(left+1, i-1) + dp(i,
28             right));
29         }
30     }
31     return mem;
32 }
33
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(NULL);
37
38     cin >> n >> s;
39     memset(tb, -1, sizeof(tb));
40     cout << dp(0, n-1) << '\n';
41
42     return 0;
43 }

```

8.8 Digit Dp

```

1 // Digit DP 1: https://atcoder.jp/contests/dp/tasks/
  dp_s
2 //
3 // find the number of integers between 1 and K (
  inclusive)

```

```

4 // where the sum of digits in base ten is a multiple
  of D
5
6 #include <bits/stdc++.h>
7
8 using namespace std;
9
10 const int MOD = 1e9+7;
11
12 string k;
13 int d;
14
15 int tb[10010][110][2];
16
17 int dp(int pos, int sum, bool under) {
18     if (pos >= k.size()) return sum == 0;
19
20     int& mem = tb[pos][sum][under];
21     if (mem != -1) return mem;
22     mem = 0;
23
24     int limit = 9;
25     if (!under) limit = k[pos] - '0';
26
27     for (int digit = 0; digit <= limit; digit++) {
28         mem += dp(pos+1, (sum + digit) % d, under | (
29         digit < limit));
30         mem %= MOD;
31     }
32     return mem;
33 }
34
35 int main() {
36     ios::sync_with_stdio(false);
37     cin.tie(NULL);
38
39     cin >> k >> d;
40
41     memset(tb, -1, sizeof(tb));
42
43     cout << (dp(0, 0, false) - 1 + MOD) % MOD << '\n'
44     ;
45
46     return 0;
47 }

```