

Competitive Programming Notebook

As Meninas Superpoderosas

Contents

1 DS	2	3.11 Flags	19
1.1 Manhattan Mst	2	3.12 Custom Unordered Map	19
1.2 Bit2d	2	3.13 Overflow	19
1.3 Bigk	3	3.14 Next Permutation	19
1.4 Mex	3	3.15 First True	20
1.5 Psum2d	3	3.16 Kosaraju	20
1.6 Dsu	4	3.17 Min Priority Queue	21
1.7 Treap	4	4 Math	21
1.8 Maxqueue	4	4.1 Is Prime	21
1.9 Segtree	5	4.2 Fft Quirino	21
1.10 Seglazystuctnode	5	4.3 Factorization	21
1.11 Seglazy	7	4.4 Sieve	21
1.12 Seghash	8	4.5 Ceil	21
1.13 Segtree Lazy Iterative	8	4.6 Log Any Base	22
1.14 Mergesorttree	9	4.7 Ifac	22
1.15 Ordered Set	10	4.8 Division Trick	22
1.16 Trie Old	10	4.9 Fexp	22
1.17 Range Color Update	10	4.10 Number Sum Product Of Divisors	22
1.18 Cht	11	4.11 Divisors	22
1.19 Bit	11	5 Graph	23
1.20 Triexor	11	5.1 Floyd Warshall	23
1.21 Querytree	12	5.2 Lca	23
1.22 Sparse	13	5.3 Bfs	23
1.23 Trie	13	5.4 Dinic	24
1.24 Kruskal	14	5.5 2sat	25
2 DP	14	5.6 Min Cost Max Flow	26
2.1 Lcs	14	5.7 Ford Fulkerson	27
2.2 Lis Binary Search	15	5.8 Dijkstra	27
2.3 Edit Distance	15	5.9 Has Negative Cycle	27
2.4 Digit Dp	15	5.10 3sat	28
2.5 Range Dp	15	6 Geometry	29
2.6 Lis Segtree	16	6.1 Convex Hull	29
2.7 Knapsack	16	7 Primitives	29
2.8 Digit Dp 2	17	7.1 Set Union Intersection	29
3 General	17	8 String	30
3.1 Last True	17	8.1 Split	30
3.2 Input By File	17	8.2 Hash	30
3.3 Mix Hash	17	8.3 Is Substring	30
3.4 Random	18	8.4 Trie Xor	31
3.5 Template	18		
3.6 Get Subsets Sum Iterative	18		
3.7 Xor Basis	18		
3.8 Xor 1 To N	18		
3.9 Base Converter	18		
3.10 Interactive	19		

1 DS

1.1 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 template <class T>
32 struct Point {
33     typedef Point P;
34     T x, y;
35     explicit Point(T x=0, T y=0) : x(x), y(y) {}
36     bool operator<(P p) const { return tie(x,y) < tie
37 (p.x,p.y); }
38     bool operator==(P p) const { return tie(x,y)==tie
39 (p.x,p.y); }
40     P operator+(P p) const { return P(x+p.x, y+p.y); }
41     P operator-(P p) const { return P(x-p.x, y-p.y); }
42     P operator*(T d) const { return P(x*d, y*d); }
43     P operator/(T d) const { return P(x/d, y/d); }
44     T dot(P p) const { return x*p.x + y*p.y; }
45     T cross(P p) const { return x*p.y - y*p.x; }
46     T cross(P a, P b) const { return (a-*this).cross(
47 b-*this); }
48     T dist2() const { return x*x + y*y; }
49     double dist() const { return sqrt((double)dist2(
50 )); }
51     // angle to x-axis in interval [-pi, pi]
52     double angle() const { return atan2(y, x); }
53     P unit() const { return *this/dist(); } // makes
54 dist()==1
55     P perp() const { return P(-y, x); } // rotates
56 +90 degrees
57     P normal() const { return perp().unit(); }
58     // returns point rotated 'a' radians ccw around
59 the origin
60     P rotate(double a) const {
61         return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a))
62     };
63     friend ostream& operator<<(ostream& os, P p) {
64         return os << "(" << p.x << "," << p.y << ")";
65     }
66 }

```

```

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 }

```

1.2 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);
36        return sum(p, q) - sum(x - 1, q) - sum(p, y -
37 1) + sum(x - 1, y - 1);
38    }
39 };

```

1.3 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, O(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 };

```

1.4 Mex

```

1 // Mex
2 //
3 // facilita queries de mex com update
4 //
5 // N eh o maior valor possivel 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 };

```

1.5 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             }
24         }
25    }
26

```

```

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 };

```

1.6 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 };

```

1.7 Treap

```

1 // treap CP algo
2
3 typedef struct item * pitem;

```

```

4 struct item {
5     int prior, value, cnt;
6     bool rev;
7     pitem l, r;
8 };
9
10 int cnt (pitem it) {
11     return it ? it->cnt : 0;
12 }
13
14 void upd_cnt (pitem it) {
15     if (it)
16         it->cnt = cnt(it->l) + cnt(it->r) + 1;
17 }
18
19 void push (pitem it) {
20     if (it && it->rev) {
21         it->rev = false;
22         swap (it->l, it->r);
23         if (it->l) it->l->rev ^= true;
24         if (it->r) it->r->rev ^= true;
25     }
26 }
27
28 void merge (pitem & t, pitem l, pitem r) {
29     push (l);
30     push (r);
31     if (!l || !r)
32         t = l ? l : r;
33     else if (l->prior > r->prior)
34         merge (l->r, l->r, r), t = l;
35     else
36         merge (r->l, l, r->l), t = r;
37     upd_cnt (t);
38 }
39
40 void split (pitem t, pitem & l, pitem & r, int key,
41 int add = 0) {
42     if (!t)
43         return void( l = r = 0 );
44     push (t);
45     int cur_key = add + cnt(t->l);
46     if (key <= cur_key)
47         split (t->l, l, t->l, key, add), r = t;
48     else
49         split (t->r, t->r, r, key, add + 1 + cnt(t->l)), l = t;
50     upd_cnt (t);
51 }
52
53 void reverse (pitem t, int l, int r) {
54     pitem t1, t2, t3;
55     split (t, t1, t2, l);
56     split (t2, t2, t3, r-l+1);
57     t2->rev ^= true;
58     merge (t, t1, t2);
59     merge (t, t, t3);
60 }
61
62 void output (pitem t) {
63     if (!t) return;
64     push (t);
65     output (t->l);
66     printf ("%d ", t->value);
67     output (t->r);
68 }

```

1.8 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     if(out.size() == 0){
21         while(in.size()){
22             ll temp = in.top().ff, ma;
23             if(out.size() == 0) ma = temp;
24             else ma = max(temp, out.top().ss);
25             out.push({temp, ma});
26             in.pop();
27         }
28     }
29     //removendo o topo de out
30     out.pop();
31 }
32
33 ll size(){
34     return in.size() + out.size();
35 }
36
37 };

```

1.9 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 = l + (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
33    //nã$ atual, intervalo na Árvore e intervalo
34    pedido

```

```

35    ll q(int x, int l, int r, int i, int j){
36        if(r < i || l > j) return 0;
37        if(l >= i && r <= j) return seg[x];
38        int mid = l + (r-l)/2;
39        return merge(q(x+x,l,mid,i,j), q(x+x+1,mid+1,
40        r,i,j));
41    }
42
43    //att posi pra val
44    void att(int x, int l, int r, int posi, ll val){
45        if(l == r){
46            seg[x] = val;
47        } else {
48            int mid = l + (r-l)/2;
49            if(posi <= mid)att(x+x,l,mid,posi,val);
50            else att(x+x+1,mid+1,r,posi,val);
51            seg[x] = merge(seg[x+x], seg[x+x+1]);
52        }
53    }
54
55    int findkth(int x, int l, int r, int k){
56        if(l == r){
57            return l;
58        } else {
59            int mid = l + (r-l)/2;
60            if(seg[x+x] >= k){
61                return findkth(x+x,l,mid,k);
62            } else {
63                return findkth(x+x+1,mid+1, r, k -
64                seg[x+x]);
65            }
66        }
67    }
68
69    ll query(int l, int r){
70        return q(1, 0, n-1, l, r);
71    }
72
73    void update(int posi, ll val){ //alterar em posi
74    pra val
75        att(1, 0, n-1, posi, val);
76    }
77
78    int findkth(int k){
79        //kth smallest, 0(logN)
80        //use position i to count how many times
81        value 'i' appear
82        //merge must be the sum of nodes
83        return findkth(1,0,n-1,k);
84    }
85
86 };

```

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         ans.suf0 = b.suf0;
71     }
72
73     ans.best0 = max({
74         a.best0,
75         b.best0,
76         a.suf0 + b.pref0
77     });
78
79     // -----
80     //
81
82     if(a.AllOne()) {
83         ans.pref1 = a.pref1 + b.pref1;
84     } else {
85         ans.pref1 = a.pref1;
86     }
87
88     if(b.AllOne()) {
89
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     return ans;
104 }
105
106 struct SegLazy {
107     private:
108
109         int n;
110         vector<Node> seg;
111         vector<bool> lazy; // precisa reverter ou nao
112
113     void build(ll x, int l, int r, string& s){
114         if(l == r){
115             int val = s[l] - '0';
116             seg[x].Init(val, l, r);
117         } else {
118             int mid = l + (r-l)/2;
119             build(x+x, l, mid, s);
120             build(x+x+1, mid+1, r, s);
121             seg[x] = Merge(seg[x+x], seg[x+x+1]);
122         }
123     }
124
125     void upd_lazy(ll node, ll l, ll r){
126
127         if(lazy[node]) {
128             seg[node].Reverse();
129         }
130
131         ll esq = node + node, dir = esq + 1;
132
133         if(dir < (int)seg.size() && lazy[node]){
134             lazy[esq] = !lazy[esq];
135             lazy[dir] = !lazy[dir];
136         }
137
138         lazy[node] = 0;
139     }
140
141     Node q(ll x, int l, int r, int i, int j){
142         upd_lazy(x,l,r);
143
144         if(r < i || l > j)
145             return Node();
146
147         if(l >= i && r <= j )
148             return seg[x];
149
150         int mid = l + (r-l)/2;
151         return Merge(q(x+x,l,mid,i,j), q(x+x+1,
152             mid+1,r,i,j));
153     }
154
155     void upd(ll x, int l, int r, int i, int j){
156         upd_lazy(x,l,r);
157         if(r < i || l > j) return;
158         if(l >= i && r <= j){

```

```

161         lazy[x] = 'lazy[x];
162         upd_lazy(x,l,r);
163     } else {
164         int mid = l + (r-1)/2;
165         upd(x+x,l,mid,i,j);
166         upd(x+x+1,mid+1,r,i,j);
167         seg[x] = Merge(seg[x+x], seg[x+x+1]);
168     }
169 }
170
171 public:
172     SegLazy(string& s){
173         n = (int)s.size();
174         seg.assign(n+n+n+n, Node());
175         lazy.assign(n+n+n+n, 0);
176         build(1,0,n-1,s);
177     }
178
179     void update(int l){
180         upd(1,0,n-1,l,l);
181     }
182
183     void update_range(int l, int r){
184         upd(1,0,n-1,l,r);
185     }
186
187     Node query(int l){
188         return q(1, 0, n-1, l, l);
189     }
190
191     Node query(int l, int r){
192         return q(1, 0, n-1, l, r);
193     }
194
195 };
196
197 void solve() {
198     int n, q;
199     string s;
200
201     cin >> n >> q >> s;
202
203     SegLazy seg(s);
204
205     while(q--) {
206         int c, l, r;
207         cin >> c >> l >> r;
208
209         if(c == 1) {
210             // inverte l...r
211             seg.update_range(l - 1, r - 1);
212         } else {
213             // query l...r
214             auto node = seg.query(l - 1, r - 1);
215             cout << node.best1 << "\n";
216         }
217     }
218 }
219
220 }
221
222 }
223
224 }

```

1.11 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,
52        r,i,j));
53    }
54
55    ll query(int l, int r){ //valor em uma posi
56        //específica -> query de [l,l];
57        return q(1, 0, n-1, l, r);
58    }
59
60    void upd(ll x, int l, int r, int i, int j, ll u){
61        upd_lazy(x,l,r);
62        if(r < i || l > j) return;
63        if(l >= i && r <= j){
64            lazy[x] += u;
65            upd_lazy(x,l,r);
66        } else {
67            int mid = l + (r-1)/2;
68            upd(x+x,l,mid,i,j,u);
69            upd(x+x+1,mid+1,r,i,j,u);
70            seg[x] = merge(seg[x+x], seg[x+x+1]);
71        }
72    }
73
74    void upd_range(int l, int r, ll u){ //intervalo e
75        //valor
76        upd(1,0,n-1,l,r,u);
77    }
78
79 };

```

1.12 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 possível que
      pode estar na estrutura
13        // isso é pra evitar que a hash
      de '0' seja igual a de '0000...'
14
15    SegtreeHash(vector<T>& s, ll P = 31, ll MOD = (1ll
      )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    //não atualiza intervalo na árvore 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,
      r, i, j), mid - max(i, l) + 1);
51    }
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-l)/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 };

```

1.13 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
      [2*i], seg[2*i+1]);
54         for (int i = 0; i < 2*n; i++) lazy[i] = 0;

```



```

55     }
56
57     ll query(int a, int b) {
58         ll ret = 0;
59         for (prop(a+=n), prop(b+=n); a <= b; ++a/=2,
60             --b/=2) {
61             if (a%2 == 1) ret = junta(ret, seg[a]);
62             if (b%2 == 0) ret = junta(ret, seg[b]);
63         }
64         return ret;
65     }
66
67     void update(int a, int b, int x) {
68         int a2 = a += n, b2 = b += n, tam = 1;
69         for (; a <= b; ++a/=2, --b/=2, tam *= 2) {
70             if (a%2 == 1) poe(a, x, tam);
71             if (b%2 == 0) poe(b, x, tam);
72         }
73         sobe(a2), sobe(b2);
74     }
75
76     int findkth(int x, int l, int r, ll k, int tam){
77         int esq = x + x;
78         int dir = x + x + 1;
79
80         upd_lazy(x, tam);
81         upd_lazy(esq, tam/2);
82         upd_lazy(dir, tam/2);
83
84         if(l == r){
85             return l;
86         } else {
87             int mid = l + (r-l)/2;
88
89             if(seg[esq] >= k){
90                 return findkth(esq, l, mid, k, tam/2);
91             } else {
92                 return findkth(dir, mid+1, r, k - seg[
93                     esq], tam/2);
94             }
95         }
96     }
97
98     int findkth(ll k){
99         // kth smallest, 0(logN)
100         // use position i to count how many times
101         // value 'i' appear
102         // merge must be the sum of nodes
103         return findkth(1, 0, n-1, k, (1 << (LOG-1)));
104     }
105 };

```

1.14 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 };

```

```

vector<int> merge(vi& a, vi& b){
    int i = 0, j = 0, p = 0;
    vi ans(a.size() + b.size());
    while(i < (int)a.size() && j < (int)b.size()){
        if(a[i] < b[j]){
            ans[p++] = a[i++];
        } else {
            ans[p++] = b[j++];
        }
    }
    while(i < (int)a.size()){
        ans[p++] = a[i++];
    }
    while(j < (int)b.size()){
        ans[p++] = b[j++];
    }
    return ans;
}

vector<ll> calc(vi& s) {
    ll sum = 0;
    vector<ll> tmp;
    for(auto &x : s) {
        sum += x;
        tmp.push_back(sum);
    }
    return tmp;
}

void seg_build(int x, int l, int r, vector<int>& s){
    if(r < l) return;
    if(l == r){
        seg[x].push_back(s[l]);
        //ps[x] = {s[l]};
    } else {
        int mid = l + (r-l)/2;
        seg_build(x+x, l, mid, s);
        seg_build(x+x+1, mid+1, r, s);
        seg[x] = merge(seg[x+x], seg[x+x+1]);
        //ps[x] = calc(seg[x]);
    }
}

//nÃs atual, intervalo na Ãrvore e intervalo
//pedido
// retorna a quantidade de numeros <= val em [l,
// r]

ll q(int x, int l, int r, int i, int j, int val){
    if(r < i || l > j) return 0;
    if(l >= i && r <= j){
        return (lower_bound(seg[x].begin(), seg[x]
        .end(), val) - seg[x].begin());
    }
    int mid = l + (r-l)/2;
    return q(x+x, l, mid, i, j, val) + q(x+x+1, mid+1,
    r, i, j, val);
}

// retorna a soma dos numeros <= val em [l, r]
// nÃs atual, intervalo na Ãrvore e intervalo
//pedido
/*
ll q(int x, int l, int r, int i, int j, ll val){
    if(r < i || l > j) return 0;
    if(l >= i && r <= j){
        auto it = upper_bound(seg[x].begin(), seg
        [x].end(), val) - seg[x].begin();
    }
}

```

```

83         if(val > seg[x].back()) {
84             return ps[x].back();
85         }
86
87         if(val < seg[x][0]) {
88             return 0;
89         }
90
91         return ps[x][it - 1];
92     }
93
94     int mid = 1 + (r-1)/2;
95     return q(x+x, l, mid, i, j, val) + q(x+x+1, mid+1,
96     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 };

```

1.15 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 //
13 // find_by_order(k) // k-th element in a set (
14 // counting from zero) -> iterator
15 //
16 // https://cses.fi/problemset/task/2169
17 //
18 // 0(log N) para insert, erase (com iterator),
19 // order_of_key, find_by_order
20
21 using namespace __gnu_pbds;
22 template <typename T>
23 using ordered_set = tree<T, null_type, less<T>,
24 rb_tree_tag, tree_order_statistics_node_update>;
25
26 void erase(ordered_set& a, int x){
27     int r = a.order_of_key(x);
28     auto it = a.find_by_order(r);
29     a.erase(it);
30 }

```

1.16 Trie Old

```

1 struct Trie {
2
3     int nxt = 1, sz, maxLet = 26; //tamanho do
4     alfabeto
5     vector< vector<int> > trie;
6     bitset<(int)1e7> finish; //modificar esse valor
7     pra ser >= n
8     //garantir que vai submeter em cpp 64
9
10     Trie(int n){
11         sz = n;
12         trie.assign(sz, vector<int>(maxLet, 0));
13     }

```

```

13 void add(string &s){
14     int cur = 0;
15     for(auto c: s){
16         //alterar esse azinho dependendo da
17         entrada!!
18         if(trie[cur][c-'a'] == 0){
19             trie[cur][c-'a'] = nxt++;
20             cur = trie[cur][c-'a'];
21         } else {
22             cur = trie[cur][c-'a'];
23         }
24     }
25     finish[cur] = 1;
26 }
27
28 int search(string& s){
29     int cur = 0;
30     for(auto c: s){
31         if(trie[cur][c - 'a'] == 0){
32             return 0;
33         }
34         cur = trie[cur][c-'a'];
35     }
36     return finish[cur];
37 }
38 };

```

1.17 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 // }
6 // query(i) retorna a cor da posicao i
7 //
8 // Complexidades (para q operacoes):
9 // update - 0(log(q)) amortizado
10 // query - 0(log(q))
11
12 template<typename T> struct color {
13     set<tuple<int, int, T>> se;
14
15     vector<tuple<int, int, T>> update(int l, int r, T
16     val) {
17         auto it = se.upper_bound({r, INF, val});
18         if (it != se.begin() and get<1>(*prev(it)) >
19         r) {
20             auto [L, R, V] = *--it;
21             se.erase(it);
22             se.emplace(L, r, V), se.emplace(r+1, R, V
23         );
24         }
25         it = se.lower_bound({l, -INF, val});
26         if (it != se.begin() and get<1>(*prev(it)) >=
27         l) {
28             auto [L, R, V] = *--it;
29             se.erase(it);
30             se.emplace(L, l-1, V), it = se.emplace(l,
31             R, V).first;
32         }
33         vector<tuple<int, int, T>> ret;
34         for (; it != se.end() and get<0>(*it) <= r;
35         it = se.erase(it))
36             ret.push_back(*it);
37         se.emplace(l, r, val);
38         return ret;
39     }
40
41     T query(int i) {
42         auto it = se.upper_bound({i, INF, T()});
43         if (it == se.begin() or get<1>(*--it) < i)
44             return -1; // nao tem

```

```

36         return get<2>(*it);
37     }
38 };

```

1.18 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 };

```

1.19 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;
42     }
43
44     /*
45     int findkth(int k){
46         //kth smallest, 0(log^2(N))
47         //use position i to count how many times
48         value 'i' appear
49         int x = 0, mx = 200;
50         for(int b = n; b > 0 && mx > 0; b /= 2){
51             while( x+b < n && query(x+b) < k && mx--
52                 > 0 ){
53                 x += b;
54             }
55         }
56         return x+1;
57     }
58 */
59 };

```

1.20 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 };

```

1.21 Querytree

```

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>(l));
19        table_max.assign(n + 5, vector<ll>(l));
20        table_min.assign(n + 5, vector<ll>(l));
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

```

```

31        in[u] = t++;
32        d[u] = depth;
33        dist[u] = sum; //sum = sum of the values in
34        edges from root to node u
35        sobe[u][0] = last; //parent of u. parent of 1
36        is 1
37        table_max[u][0] = we;
38        table_min[u][0] = we;
39        for(auto v: arr[u]) if(v.ff != last){
40            Euler_Tour(v.ff, u, v.ss, depth + 1, sum
41            + v.ss);
42        }
43        out[u] = t++;
44    }
45
46    void build_table(){ //binary lifting
47        assert(euler);
48        build = 1; //remeber use this function before
49        queries
50        for(int k = 1; k < l; k++){
51            for(int i = 1; i <= n; i++){
52                sobe[i][k] = sobe[sobe[i][k-1]][k-1];
53                table_max[i][k] = max(table_max[i][k
54                - 1], table_max[sobe[i][k-1]][k-1]);
55                table_min[i][k] = min(table_min[i][k
56                - 1], table_min[sobe[i][k-1]][k-1]);
57            }
58        }
59    }
60
61    int is_ancestor(int u, int v){ // return 1 if u
62    is ancestor of v
63        assert(euler);
64        return in[u] <= in[v] && out[u] >= out[v];
65    }
66
67    int lca(int u, int v){ //return lca of u and v
68        assert(build && euler);
69        if(is_ancestor(u, v)) return u;
70        if(is_ancestor(v, u)) return v;
71        int lca = u;
72        for(int k = l - 1; k >= 0; k--){
73            int tmp = sobe[lca][k];
74            if(!is_ancestor(tmp, v)){
75                lca = tmp;
76            }
77        }
78        return sobe[lca][0];
79    }
80
81    int lca(int u, int v, int root) { //return lca of
82    u and v when tree is rooted at 'root'
83        return lca(u, v) ^ lca(v, root) ^ lca(root, u
84        ); //magic
85    }
86
87    int up_k(int u, int qt){ //return node k levels
88    higher starting from u
89        assert(build && euler);
90        for(int b = 0; b < l; b++){
91            if(qt%2) u = sobe[u][b];
92            qt >>= 1;
93        }
94        return u;
95    }
96
97    ll goUpMax(int u, int to){ //return the max
98    weigth of a edge going from u to 'to'
99        assert(build);
100        if(u == to) return 0;
101        ll mx = table_max[u][0];
102        for(int k = l - 1; k >= 0; k--){
103            int tmp = sobe[u][k];

```

```

93         if( !is_ancestor(tmp, to) ){
94             mx = max(mx, table_max[u][k]);
95             u = tmp;
96         }
97     }
98     return max(mx, table_max[u][0]);
99 }
100
101 ll max_edge(int u, int v){ //return the max
102     weight of a edge in the simple path from u to v
103     assert(build);
104     int ancestor = lca(u, v);
105     ll a = goUpMax(u, ancestor), b = goUpMax(v,
106     ancestor);
107     if(ancestor == u) return b;
108     else if(ancestor == v) return a;
109     return max(a,b);
110 }
111
112 ll goUpMin(int u, int to){ //return the min
113     weight of a edge going from u to 'to'
114     assert(build);
115     if(u == to) return oo;
116     ll mx = table_min[u][0];
117     for(int k = 1 - 1; k >= 0; k--){
118         int tmp = sobe[u][k];
119         if( !is_ancestor(tmp, to) ){
120             mx = min(mx, table_min[u][k]);
121             u = tmp;
122         }
123     }
124     return min(mx, table_min[u][0]);
125 }
126
127 ll min_edge(int u, int v){ //return the min
128     weight of a edge in the simple path from u to v
129     assert(build);
130     int ancestor = lca(u, v);
131     ll a = goUpMin(u, ancestor), b = goUpMin(v,
132     ancestor);
133     if(ancestor == u) return b;
134     else if(ancestor == v) return a;
135     return min(a,b);
136 }
137
138 ll query_dist(int u, int v){ //distance of nodes
139     u and v
140     int x = lca(u, v);
141     return dist[u] - dist[x] + dist[v] - dist[x];
142 }
143
144 int kth_between(int u, int v, int k){ //kth node
145     in the simple path from u to v; if k = 1, ans = u
146     k--;
147     int x = lca(u, v);
148     if( k > d[u] - d[x] ){
149         k -= (d[u] - d[x]);
150         return up_k(v, d[v]-d[x]-k);
151     }
152     return up_k(u, k);
153 }
154
155 int main() {
156     ios::sync_with_stdio(false);
157     cin.tie(NULL);
158
159     int t = 1, n, u, v, w, k;
160     string s;
161     cin >> t;
162     while(t--){
163         cin >> n;

```

```

159     QueryTree arr(n);
160     for(int i = 1; i < n; i++){
161         cin >> u >> v >> w;
162         arr.add_edge(u,v,w);
163     }
164     arr.Euler_Tour(1);
165     arr.build_table();
166     while(cin >> s, s != "DONE"){
167         cin >> u >> v;
168         if(s == "DIST") {
169             cout << arr.query_dist(u, v) << "\n";
170         } else {
171             cin >> k;
172             cout << arr.kth_between(u,v,k) << "\n";
173         }
174     }
175     cout << "\n";
176 }
177
178 }

```

1.22 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
12    Sparse(vector<int>& v){ //Constrói a tabela
13        int n = v.size(), logn = 0;
14        while((1<<logn) <= n) logn++;
15        arr.assign(n, vector<int>(logn, 0));
16        for(int i = 0; i < n; i++){
17            arr[i][0] = v[i];
18            for(int k = 1; k < logn; k++){
19                for(int i = 0; i < n; i++){
20                    if(i + ( 1 << k) - 1 >= n)
21                        break;
22                    int p = i + ( 1 << (k-1) );
23                    arr[i][k] = op( arr[i][ k-1 ] , arr[p
24                    ][k-1] );
25                }
26            }
27        }
28
29        int query(int l, int r){
30            int pot = 31 - __builtin_clz(r-l+1); //r-l+1
31            sãO INTEIROS, nãO ll
32            int k = (1 << pot) ;
33            return op( arr[l][pot] , arr[ r - (k-1) ][
34            pot] );
35        }
36    };

```

1.23 Trie

```

1 struct Trie {
2
3     struct Node {
4         map<char, Node> adj; // dã pra trocar por
5         vector(26)
6         ll finishHere;
7
8         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 }
69
70 void debug() {
71     debugRec(mainNode, 0);
72 }
73 };

```

1.24 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    };
15
16    vector<Edge> kruskal(vector<Edge> edges, int n) {
17        vector<Edge> result;
18        ll cost = 0;
19
20        sort(edges.begin(), edges.end());
21        DSU dsu(n);
22
23        for (auto e : edges) {
24            if (!dsu.same(e.u, e.v)) {
25                cost += e.weight;
26                result.push_back(e);
27                dsu.unite(e.u, e.v);
28            }
29        }
30
31        return result;
32    }

```

2 DP

2.1 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
8 // resposta
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     }

```

```

39 // recuperar resposta
40
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 }

```

2.2 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 }

```

2.3 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 }

```

```

29
30     return ans;
31 }

```

2.4 Digit Dp

```

1 // Digit DP 1: https://atcoder.jp/contests/dp/tasks/
2 // dp_s
3 // find the number of integers between 1 and K (
4 // inclusive)
5 // where the sum of digits in base ten is a multiple
6 // of D
7
8 #include <bits/stdc++.h>
9
10 using namespace std;
11
12 const int MOD = 1e9+7;
13
14 string k;
15 int d;
16
17 int tb[10010][110][2];
18
19 int dp(int pos, int sum, bool under) {
20     if (pos >= k.size()) return sum == 0;
21
22     int& mem = tb[pos][sum][under];
23     if (mem != -1) return mem;
24     mem = 0;
25
26     int limit = 9;
27     if (!under) limit = k[pos] - '0';
28
29     for (int digit = 0; digit <= limit; digit++) {
30         mem += dp(pos+1, (sum + digit) % d, under | (
31         digit < limit));
32         mem %= MOD;
33     }
34
35     return mem;
36 }
37
38 int main() {
39     ios::sync_with_stdio(false);
40     cin.tie(NULL);
41
42     cin >> k >> d;
43
44     memset(tb, -1, sizeof(tb));
45
46     cout << (dp(0, 0, false) - 1 + MOD) % MOD << '\n'
47     ;
48
49     return 0;
50 }

```

2.5 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
11
12 #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 a 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 }

```

2.6 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';

```

2.7 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 sãr preciso do resultado
5     //final.
6     //O(W) em memória
7     vector<vector<ll>> table(2, vector<ll>(W + 1, 0))
8     ;
9     int n = (int)value.size();

```

```

7
8     for(int k = 1; k <= n; k++) {
9         for(int i = 0; i <= W; i++) {
10             if(i - weight[k - 1] >= 0) {
11                 table[k % 2][i] = max(table[(k - 1) %
12                 2][i],
13                     value[k - 1] + table[(k - 1) %
14                     2][i - weight[k - 1]]);
15             } else {
16                 table[k % 2][i] = max(table[(k - 1) %
17                 2][i], table[k % 2][i]);
18             }
19         }
20     }
21     return table[n % 2][W];
22 }
23
24 ll knapsack(vector<ll>& weight, vector<ll>& value,
25     int W) {
26     //Usar essa knapsack se, em algum momento,
27     //precisar recuperar os índices
28     //O(NW) em memória
29
30     int n = (int)value.size();
31     vector<vector<ll>> table(W + 1, vector<ll>(n + 1,
32     0));
33
34     for(int k = 1; k <= n; k++) {
35         for(int i = 0; i <= W; i++) {
36             if(i - weight[k - 1] >= 0) {
37                 table[i][k] = max(table[i][k - 1],
38                     value[k - 1] + table[i - weight[k
39                     - 1]][k - 1]);
40             } else {
41                 table[i][k] = max(table[i][k - 1],
42                     table[i][k]);
43             }
44         }
45     }
46
47     /*
48     int per = W;
49     vector<int> idx;
50     for(int k = n; k > 0; k--) {
51         if(table[per][k] == table[per][k - 1]){
52             continue;
53         } else {
54             idx.push_back(k - 1);
55             per -= weight[k - 1];
56         }
57     }
58     */
59     return table[W][n];
60 }
61
62 const int MOD = 998244353;
63
64 struct Knapsack {
65
66     int S; // max value
67     vector<ll> dp;
68
69     Knapsack(int S_) {
70         S = S_ + 5;
71         dp.assign(S, 0);
72         dp[0] = 1;
73     }
74
75     void Add(int val) {
76         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 void solve() {
95     int n, w;
96     cin >> n >> w;
97     vector<ll> weight(n), value(n);
98     for(int i = 0; i < n; i++) {
99         cin >> weight[i] >> value[i];
100     }
101     cout << knapsack(weight, value, w) << "\n";
102 }
103
104 }

```

2.8 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
    under, bool started) {
16     if (pos >= (int)number.size()) {
17         return 1;
18     }
19
20     ll& mem = tb[pos][last_digit][under][started];
21     if (mem != -1) return mem;
22     mem = 0;
23
24     int limit = 9;
25     if (!under) limit = number[pos] - '0';
26
27     for (int digit = 0; digit <= limit; digit++) {
28         if (started && digit == last_digit) continue;
29
30         bool is_under = under || (digit < limit);
31         bool is_started = started || (digit != 0);
32
33         mem += dp(number, pos+1, digit, is_under,
    is_started);
34     }

```

```

35
36     return mem;
37 }
38
39 ll solve(ll ubound) {
40     memset(tb, -1, sizeof(tb));
41     string number = to_string(ubound);
42     return dp(number, 0, 10, 0, 0);
43 }
44
45 int main() {
46     ios::sync_with_stdio(false);
47     cin.tie(NULL);
48
49     ll a, b; cin >> a >> b;
50     cout << solve(b) - solve(a-1) << '\n';
51
52     return 0;
53 }

```

3 General

3.1 Last True

```

1 // Binary Search (last_true)
2
3 // last_true(2, 10, [](int x) { return x * x <= 30;
4 // }); // outputs 5
5 //
6 // [1, 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 }

```

3.2 Input By File

```

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

```

3.3 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))
    ;}

```

3.4 Random

```

1 int main() {
2     ios::sync_with_stdio(false);
3     cin.tie(NULL);
4
5     //mt19937 rng(chrono::steady_clock::now().
6     time_since_epoch().count()); //gerar int
7     mt19937_64 rng(chrono::steady_clock::now().
8     time_since_epoch().count()); //gerar ll
9
10    /*usar rng() pra gerar numeros aleat rios.*/
11    /*usar rng() % x pra gerar numeros em [0, x-1]*/
12    for(int i = 0; i < 10; i++){
13        cout << rng() << endl;
14    }
15    vector<ll> arr = {1,2,3,4,5,6,7,8,9};
16    /*d   pra usar no shuffle de vector tamb  m*/
17    shuffle(arr.begin(), arr.end(),rng);
18    for(auto &x: arr)
19        cout << x << endl;
20 }

```

3.5 Template

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(NULL);
8
9
10    return 0;
11 }

```

3.6 Get Subsets Sum Iterative

```

1 vector<ll> get_subset_sums(int l, int r, vector<ll>&
2 arr) {
3     vector<ll> ans;
4
5     int len = r-l+1;
6     for (int i = 0; i < (1 << len); i++) {
7         ll sum = 0;
8
9         for (int j = 0; j < len; j++) {
10             if (i&(1 << j)) {
11                 sum += arr[l + j];
12             }
13         }
14         ans.push_back(sum);
15     }
16
17     return ans;
18 }

```

3.7 Xor Basis

```

1 // XOR Basis
2 // You are given a set of $N$ integer values. You
3 // should find the minimum number of values that you
4 // need to add to the set such that the following
5 // will hold true:
6 // For every two integers $A$ and $B$ in the set,
7 // their bitwise xor $A \oplus B$ is also in the set
8 .
9
10 vector<ll> basis;
11
12 void add(ll x) {
13     for (int i = 0; i < (int)basis.size(); i++) {
14         // reduce x using the current basis vectors
15         x = min(x, x ^ basis[i]);
16     }
17
18     if (x != 0) { basis.push_back(x); }
19 }
20
21 ll res = (1LL << (int)basis.size()) - n;

```

3.8 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 }

```

3.9 Base Converter

```

1 const string digits = "0123456789
2 ABCDEFGHIJKLMNOPQRSTUVWXYZ";
3
4 ll tobase10(string number, int base) {
5     map<char, int> val;
6     for (int i = 0; i < digits.size(); i++) {
7         val[digits[i]] = i;
8     }
9
10    ll ans = 0, pot = 1;
11
12    for (int i = number.size() - 1; i >= 0; i--) {
13        ans += val[number[i]] * pot;
14        pot *= base;
15    }
16
17    return ans;
18 }
19
20 string frombase10(ll number, int base) {
21     if (number == 0) return "0";
22
23     string ans = "";
24
25     while (number > 0) {
26         ans += digits[number % base];
27         number /= base;
28     }
29
30    reverse(ans.begin(), ans.end());
31
32    return ans;
33 }

```

```

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 }

```

3.10 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 }

```

3.11 Flags

```

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

```

3.12 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::
steady_clock::now().time_since_epoch().count();
14        return splitmix64(x + FIXED_RANDOM);
15    }
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) ^
2038074743 ^ a.second;
24     }
25 };

```

3.13 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)

```

3.14 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()));

```

3.15 First True

```

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

```

3.16 Kosaraju

```

1 // https://codeforces.com/blog/entry/125435
2 #ifdef MAXWELL_LOCAL_DEBUG
3 #include "debug/debug_template.cpp"
4 #define dbg debug
5 #else
6 #define debug(...)
7 #define dbg debug
8 #define debugArr(arr, n)
9 #endif
10
11 #include <bits/stdc++.h>
12 #define ff first
13 #define ss second
14
15 using namespace std;
16 using ll = long long;
17 using ld = long double;
18 using pii = pair<int, int>;
19 using vi = vector<int>;
20
21 using tii = tuple<int, int, int>;
22 // auto [a,b,c] = ...
23 // .insert({a,b,c})
24
25 const int oo = (int)1e9 + 5; //INF to INT
26 const ll OO = 0x3f3f3f3f3f3f3f3fLL; //INF to LL
27
28 struct Kosaraju {
29
30     int N;
31     int cntComps;
32
33     vector<vector<int>> g;
34     vector<vector<int>> gi;
35

```

```

36     stack<int> S;
37     vector<int> vis;
38     vector<int> comp;
39
40     Kosaraju(vector<vector<int>>& arr) {
41         N = (int)arr.size();
42         cntComps = 0;
43
44         g.resize(N);
45         gi.resize(N);
46         vis.resize(N);
47         comp.resize(N);
48
49         for(int i = 0; i < (int)arr.size(); i++) {
50             for(auto &v : arr[i]) {
51                 g[i].push_back(v);
52                 gi[v].push_back(i);
53             }
54         }
55
56         run();
57     }
58
59     void dfs(int u) {
60         vis[u] = 1;
61         for(auto &v : g[u]) if(!vis[v]) {
62             dfs(v);
63         }
64         S.push(u);
65     }
66
67     void scc(int u, int c) {
68         vis[u] = 1;
69         comp[u] = c;
70         for(auto &v : gi[u]) if(!vis[v]) {
71             scc(v, c);
72         }
73     }
74
75     void run() {
76         vis.assign(N, 0);
77
78         for(int i = 0; i < N; i++) if(!vis[i]) {
79             dfs(i);
80         }
81
82         vis.assign(N, 0);
83
84         while((int)S.size()) {
85             int u = S.top();
86             S.pop();
87             if(!vis[u]) {
88                 scc(u, cntComps++);
89             }
90         }
91     }
92
93 };
94
95 int main() {
96     ios::sync_with_stdio(false);
97     cin.tie(NULL);
98
99     int t = 1;
100
101     while(t--) {
102         solve();
103     }
104
105 }
106 }

```

3.17 Min Priority Queue

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

4 Math

4.1 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 }
```

4.2 Fft Quirino

```
1 // FFT
2 //
3 // boa em memÓria 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>
  > const& B) {
44     vector<cd> fa(begin(A), end(A)), fb(begin(B), end(B));
45     int N = 1;
```

```
46     while (N < size(A) + size(B))
47         N <= 1;
48     fa.resize(N);
49     fb.resize(N);
50
51     fft(fa, false);
52     fft(fb, false);
53     for (int i = 0; i < N; i++)
54         fa[i] *= fb[i];
55     fft(fa, true);
56
57     vector<int> result(N);
58     for (int i = 0; i < N; i++)
59         result[i] = round(fa[i].real());
60     return result;
61 }
```

4.3 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 }
```

4.4 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    }
```

4.5 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 }

```

4.6 Log Any Base

```

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

```

4.7 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 }

```

4.8 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 }

```

4.9 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 }

```

4.10 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
   as a prime factorization.
4 //
5 // Input:
6 // The first line has an integer n: the number of
   parts in the prime factorization.
7 // After this, there are n lines that describe the
   factorization. Each line has two numbers x and k
   where x is a prime and k is its power.
8 //
9 // Output:
10 // Print three integers modulo 10^9+7: the number,
   sum and product of the divisors.
11 //
12 // Constraints:
13 // (1 <= n <= 1e5) ; (2 <= x <= 1e6) ; (1 <= k <= 1e9
   ) ; each x is a distinct prime
14
15 #include <bits/stdc++.h>
16 typedef long long ll;
17 using namespace std;
18
19 const ll MOD = 1e9 + 7;
20
21 ll expo(ll base, ll pow) {
22     ll ans = 1;
23     while (pow) {
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,
   div_cnt2 = 1;
39     for (int i = 0; i < n; i++) {
40         div_cnt = div_cnt * (k[i] + 1) % MOD;
41         div_sum = div_sum * (expo(p[i], k[i] + 1) -
   1) % MOD *
42             expo(p[i] - 1, MOD - 2) % MOD;
43         div_prod = expo(div_prod, k[i] + 1) *
44             expo(expo(p[i], (k[i] * (k[i] + 1)
   / 2)), div_cnt2) % MOD;
45         div_cnt2 = div_cnt2 * (k[i] + 1) % (MOD - 1);
46     }
47     cout << div_cnt << ' ' << div_sum << ' ' <<
   div_prod;
48     return 0;
49 }

```

4.11 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 }

```

5 Graph

5.1 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 }

```

5.2 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 =
17         1) {
18         up.assign(n+1, vector<int>(MAXE, -1));
19         dep.assign(n+1, 0);
20
21         dep[root] = 1;
22         dfs(root, -1, adj);
23
24         for (int j = 1; j < MAXE; j++) {
25             for (int i = 1; i <= n; i++) {
26                 if (up[i][j-1] != -1)
27                     up[i][j] = up[ up[i][j-1] ][j-1];
28             }
29         }
30
31         void dfs(int x, int p, vector<vector<int>>& adj)
32         {

```

```

32         up[x][0] = p;
33         for (auto e : adj[x]) {
34             if (e != p) {
35                 dep[e] = dep[x] + 1;
36                 dfs(e, x, adj);
37             }
38         }
39     }
40
41     int jump(int x, int k) { // jump from node x k
42         times
43         for (int i = 0; i < MAXE; i++) {
44             if (k && (1 << i) && x != -1) x = up[x][i];
45         }
46         return x;
47     }
48
49     int lca(int a, int b) {
50         if (dep[a] > dep[b]) swap(a, b);
51         b = jump(b, dep[b] - dep[a]);
52
53         if (a == b) return a;
54
55         for (int i = MAXE-1; i >= 0; i--) {
56             if (up[a][i] != up[b][i]) {
57                 a = up[a][i];
58                 b = up[b][i];
59             }
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     };

```

5.3 Bfs

```

1  vector<vector<int>> adj; // adjacency list
2  representation
3  int n; // number of nodes
4  int s; // source vertex
5
6  queue<int> q;
7  vector<bool> used(n + 1);
8  vector<int> d(n + 1), p(n + 1);
9
10 q.push(s);
11 used[s] = true;
12 p[s] = -1;
13 while (!q.empty()) {
14     int v = q.front();
15     q.pop();
16     for (int u : adj[v]) {
17         if (!used[u]) {
18             used[u] = true;
19             q.push(u);
20             d[u] = d[v] + 1;
21             p[u] = v;
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 reverse(path.begin(), path.end());
34
35 cout << "Path: ";
36 for (int v : path)
37     cout << v << " ";
38 }

```

5.4 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(to), cap(cap), flow(0) {};
```

```

20
21     ll remaining_cap() {
22         return cap - flow;
23     }
24
25     void augment(ll bottle_neck) {
26         flow += bottle_neck;
27         residual->flow -= bottle_neck;
28     }
29
30     bool is_residual() {
31         return cap == 0;
32     }
33 };
34
35 struct Dinic {
36     int n;
37     vector<vector<Edge*>> adj;
38     vector<int> level, next;
39
40     Dinic(int n): n(n) {
41         adj.assign(n+1, vector<Edge*>());
42         level.assign(n+1, -1);
43         next.assign(n+1, 0);
44     }
45
46     void add_edge(int from, int to, ll cap) {
47         auto e1 = new Edge(from, to, cap);
48         auto e2 = new Edge(to, from, 0);
49
50         e1->residual = e2;
51         e2->residual = e1;
52
53         adj[from].push_back(e1);
54         adj[to].push_back(e2);
55     }
56
57     bool bfs(int s, int t) {
58         fill(level.begin(), level.end(), -1);
59         queue<int> q;
60
61         q.push(s);
62         level[s] = 1;

```

```

63         while (q.size()) {
64             int curr = q.front();
65             q.pop();
66
67             for (auto edge : adj[curr]) {
68                 if (edge->remaining_cap() > 0 &&
69                     level[edge->to] == -1) {
70                     level[edge->to] = level[curr] +
71                     1;
72                     q.push(edge->to);
73                 }
74             }
75
76             return level[t] != -1;
77         }
78
79         ll dfs(int x, int t, ll flow) {
80             if (x == t) return flow;
81
82             for (int& cid = next[x]; cid < (int)adj[x].
83                 size(); cid++) {
84                 auto& edge = adj[x][cid];
85                 ll cap = edge->remaining_cap();
86
87                 if (cap > 0 && level[edge->to] == level[x]
88                     + 1) {
89                     ll sent = dfs(edge->to, t, min(flow,
90                         cap)); // bottle neck
91                     if (sent > 0) {
92                         edge->augment(sent);
93                         return sent;
94                     }
95                 }
96             }
97
98             return 0;
99         }
100
101         ll solve(int s, int t) {
102             ll max_flow = 0;
103
104             while (bfs(s, t)) {
105                 fill(next.begin(), next.end(), 0);
106
107                 while (ll sent = dfs(s, t, FLOW_INF)) {
108                     max_flow += sent;
109                 }
110             }
111
112             return max_flow;
113         }
114
115         // path recover
116         vector<bool> vis;
117         vector<int> curr;
118
119         bool dfs2(int x, int& t) {
120             vis[x] = true;
121             bool arrived = false;
122
123             if (x == t) {
124                 curr.push_back(x);
125                 return true;
126             }
127
128             for (auto e : adj[x]) {
129                 if (e->flow > 0 && !vis[e->to]) { // !e->
130                     is_residual() &&
131                     bool aux = dfs2(e->to, t);
132
133                     if (aux) {

```



```

130         arrived = true;
131         e->flow--;
132     }
133 }
134 }
135
136 if (arrived) curr.push_back(x);
137
138 return arrived;
139 }
140
141 vector<vector<int>> get_paths(int s, int t) {
142     vector<vector<int>> ans;
143
144     while (true) {
145         curr.clear();
146         vis.assign(n+1, false);
147
148         if (!dfs2(s, t)) break;
149
150         reverse(curr.begin(), curr.end());
151         ans.push_back(curr);
152     }
153
154     return ans;
155 }
156 };

```

5.5 2sat

```

1 // 2SAT
2 //
3 // verifica se existe e encontra solu~ão
4 // para fórmulas 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 não
15 // foram testadas)
16 //
17 // O(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;
24     stack<int> topo; // ordem topológica
25
26     sat() {}
27     sat(int n_) : n(n_), tot(n), adj(2*n), adjt(2*n) {}
28
29     void dfs(int x) {
30         vis[x] = true;
31
32         for (auto e : adj[x]) {
33             if (!vis[e]) dfs(e);
34         }
35
36         topo.push(x);
37     }
38
39     void dfst(int x, int& id) {
40         vis[x] = true;
41         comp[x] = id;

```

```

42         for (auto e : adjt[x]) {
43             if (!vis[e]) dfst(e, id);
44         }
45     }
46
47     void add_impl(int a, int b) { // a -> b = (!a or 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 b)
63         add_or(~a, b), add_or(a, ~b), add_or(~a, ~b);
64     }
65
66     void add_and(int a, int b) { // a and b
67         add_or(a, b), add_or(~a, b), add_or(a, ~b);
68     }
69
70     void add_nand(int a, int b) { // a nand b = !(a and b)
71         add_or(~a, ~b);
72     }
73
74     void add_xor(int a, int b) { // a xor b = (a != b)
75         add_or(a, b), add_or(~a, ~b);
76     }
77
78     void add_xnor(int a, int b) { // a xnor b = !(a xor b) = (a = b)
79         add_xor(~a, b);
80     }
81
82     void add_true(int a) { // a = T
83         add_or(a, ~a);
84     }
85
86     void add_false(int a) { // a = F
87         add_and(a, ~a);
88     }
89
90     // magia - brunomaletta
91     void add_true_old(int a) { // a = T (n sei se funciona)
92         add_impl(~a, a);
93     }
94
95     void at_most_one(vector<int> v) { // no max um verdadeiro
96         adj.resize(2*(tot+v.size()));
97         for (int i = 0; i < v.size(); i++) {
98             add_impl(tot+i, ~v[i]);
99             if (i) {
100                 add_impl(tot+i, tot+i-1);
101                 add_impl(v[i], tot+i-1);
102             }
103         }
104         tot += v.size();
105     }
106
107     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 };

```

5.6 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
6 // Se for um dag, da pra substituir o SPFA por uma DP
7 // pagar O(nm) no comeco
8 // Se nao tiver aresta com custo negativo, nao
9 // precisa do SPFA
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
15         reversa, fluxo, capacidade
16         bool res; // se eh reversa
17         T cost; // custo da unidade de fluxo
18         edge() : to(0), rev(0), flow(0), cap(0), cost
19         (0), res(false) {}
20         edge(int to_, int rev_, int flow_, int cap_,
21             T cost_, bool res_)
22             : to(to_), rev(rev_), flow(flow_), cap(
23             cap_), res(res_), cost(cost_) {}
24     };
25     vector<vector<edge>> g;
26     vector<int> par_idx, par;
27     T inf;
28     vector<T> dist;
29
30     mcmf(int n) : g(n), par_idx(n), par(n), inf(
numeric_limits<T>::max()/3) {}
31
32     void add(int u, int v, int w, T cost) { // de u
33     pra v com cap w e custo cost
34         edge a = edge(v, g[v].size(), 0, w, cost,
35         false);

```

```

36         edge b = edge(u, g[u].size(), 0, 0, -cost,
37         true);
38         g[u].push_back(a);
39         g[v].push_back(b);
40     }
41
42     vector<T> spfa(int s) { // nao precisa se nao
43     tiver custo negativo
44         deque<int> q;
45         vector<bool> is_inside(g.size(), 0);
46         dist = vector<T>(g.size(), inf);
47
48         dist[s] = 0;
49         q.push_back(s);
50         is_inside[s] = true;
51
52         while (!q.empty()) {
53             int v = q.front();
54             q.pop_front();
55             is_inside[v] = false;
56
57             for (int i = 0; i < g[v].size(); i++) {
58                 auto [to, rev, flow, cap, res, cost]
59                 = g[v][i];
60                 if (flow < cap and dist[v] + cost <
61                 dist[to]) {
62                     dist[to] = dist[v] + cost;
63
64                     if (is_inside[to]) continue;
65                     if (!q.empty() and dist[to] >
66                     dist[q.front()]) q.push_back(to);
67                     else q.push_front(to);
68                     is_inside[to] = true;
69                 }
70             }
71             return dist;
72         }
73     }
74
75     bool dijkstra(int s, int t, vector<T>& pot) {
76         priority_queue<pair<T, int>, vector<pair<T,
77         int>>, greater<>> q;
78         dist = vector<T>(g.size(), inf);
79         dist[s] = 0;
80         q.emplace(0, s);
81         while (q.size()) {
82             auto [d, v] = q.top();
83             q.pop();
84             if (dist[v] < d) continue;
85             for (int i = 0; i < g[v].size(); i++) {
86                 auto [to, rev, flow, cap, res, cost]
87                 = g[v][i];
88                 cost += pot[v] - pot[to];
89                 if (flow < cap and dist[v] + cost <
90                 dist[to]) {
91                     dist[to] = dist[v] + cost;
92                     q.emplace(dist[to], to);
93                     par_idx[to] = i, par[to] = v;
94                 }
95             }
96             return dist[t] < inf;
97         }
98     }
99
100     pair<int, T> min_cost_flow(int s, int t, int flow
101     = INF) {
102         vector<T> pot(g.size(), 0);
103         pot = spfa(s); // mudar algoritmo de caminho
104         minimo aqui
105
106         int f = 0;
107         T ret = 0;
108         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
101                [u]][par_idx[u]].flow);
102             u = par[u];
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
110             -= mn_flow;
111             u = par[u];
112         }
113         f += mn_flow;
114     }
115
116     return make_pair(f, ret);
117 }
118
119 // Opcional: retorna as arestas originais por
120 // onde passa flow = cap
121 vector<pair<int,int>> recover() {
122     vector<pair<int,int>> used;
123     for (int i = 0; i < g.size(); i++) for (edge
124     e : g[i])
125         if (e.flow == e.cap && !e.res) used.
126         push_back({i, e.to});
127     return used;
128 }

```

5.7 Ford Fulkerson

```

1 // Ford-Fulkerson
2 //
3 // max-flow / min-cut
4 //
5 // MAX nÃs
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++) {

```

```

        if (adj[x][i] > 0 && !used[i]) {
            ll sent = dfs(i, t, min(amount, adj[x
            ][i]));
        }
        if (sent > 0) {
            adj[x][i] -= sent;
            adj[i][x] += sent;
            return sent;
        }
    }
    return 0;
}

ll max_flow(int s, int t) { // source and sink
    ll total = 0;
    ll sent = -1;
    while (sent != 0) {
        memset(used, 0, sizeof(used));
        sent = dfs(s, t, INT_MAX);
        total += sent;
    }
    return total;
}
};

```

5.8 Dijkstra

```

1 const int INF = 1e9+17;
2 vector<vector<pair<int, int>>> adj; // {neighbor,
3     weight}
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());
16
17        for (auto edge : adj[v]) {
18            int to = edge.first;
19            int len = edge.second;
20
21            if (d[v] + len < d[to]) {
22                q.erase({d[to], to});
23                d[to] = d[v] + len;
24                p[to] = v;
25                q.insert({d[to], to});
26            }
27        }
28    }
}

```

5.9 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 {

```

```

7     const int INF { 1e9+17 };
8
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
27    return false;

```

5.10 3sat

```

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

```

```

49 void push(int u) {
50     x[u + n] = 1;
51     decision_stack.back().push_back(u);
52     for (auto i: g[u + n]) if (pos[i]++ == 0) {
53         for (auto u: lit[i]) --occ[u+n];
54     }
55     for (auto i: g[~u + n]) {
56         ++neg[i];
57         if (pos[i] == 0) unit_stack.push_back(i);
58     }
59 }
60 void pop() {
61     int u = decision_stack.back().back();
62     decision_stack.back().pop_back();
63     x[u + n] = 0;
64     for (auto i: g[u + n]) if (--pos[i] == 0) {
65         for (auto u: lit[i]) ++occ[u + n];
66     }
67     for (auto i: g[~u+n]) --neg[i];
68 }
69 bool reduction() {
70     while(!unit_stack.empty() || !pure_stack.empty())
71     {
72         if(!pure_stack.empty()) { // pure literal
73             elimination
74             int u = pure_stack.back();
75             pure_stack.pop_back();
76             if (occ[u + n] == 1 && occ[~u + n] == 0) push
77             (u);
78         } else { // unit propagation
79             int i = unit_stack.back();
80             unit_stack.pop_back();
81             if(pos[i] > 0) continue;
82             if(neg[i] == lit[i].size()) return false;
83             if(neg[i] + 1 == lit[i].size()) {
84                 int w = n;
85                 for (int u: lit[i]) if (!x[u + n] && !x[~u
86                 + n]) w = u;
87                 if (x[~w + n]) return false;
88                 push(w);
89             }
90         }
91     }
92     return true;
93 }
94 bool ok() {
95     x.assign(2*n,0);
96     pos = neg = vector<int>(lit.size());
97     decision_stack.assign(1, {});
98     while(1) {
99         if(reduction()) {
100             int s = 0;
101             for(int u = 0; u < n; ++u) if(occ[s + n] +
102             occ[~s + n] < occ[u + n] + occ[~u + n]) s = u;
103             if(occ[s + n] + occ[~s + n] == 0) return true
104             ;
105             decision_stack.push_back({});
106             push(s);
107         } else {
108             int s = decision_stack.back()[0];
109             while(!decision_stack.back().empty()) pop();
110             decision_stack.pop_back();
111             if (decision_stack.empty()) return false;
112             push(~s);
113         }
114     }
115 }
116 };
117
118 int32_t main() {
119     int n = 9;
120     SAT_GOD t(n);
121     t.add(0, 0, 1, 1);

```

```

112 t.add(1, 0);
113 t.add(1, 0, 3, 1, 5, 1);
114 cout << t.ok() << endl;
115 }

```

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]};

```

```

59     for (int i = 2; i < n; i++) {
60         upper_hull.push_back(points[i]);
61
62         int sz = upper_hull.size();
63
64         while (sz >= 3 && dir(upper_hull[sz-3],
65         upper_hull[sz-2], upper_hull[sz-1]) == -1) {
66             upper_hull.pop_back();
67             upper_hull.push_back(points[i]);
68             sz--;
69         }
70     }
71
72     vector<Point> lower_hull = {points[n-1], points[n
73     -2]};
74     for (int i = n-3; i >= 0; i--) {
75         lower_hull.push_back(points[i]);
76
77         int sz = lower_hull.size();
78
79         while (sz >= 3 && dir(lower_hull[sz-3],
80         lower_hull[sz-2], lower_hull[sz-1]) == -1) {
81             lower_hull.pop_back();
82             lower_hull.push_back(points[i]);
83             sz--;
84         }
85     }
86
87     // reverse(lower_hull.begin(), lower_hull.end());
88     // counterclockwise
89
90     for (int i = (int)lower_hull.size() - 2; i > 0; i
91     --) {
92         upper_hull.push_back(lower_hull[i]);
93     }
94
95     return upper_hull;
96 }

```

7 Primitives

7.1 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
8     , CMP, ALLOC> &s2)
9 {
10     std::set<T, CMP, ALLOC> s;
11     std::set_intersection(s1.begin(), s1.end(), s2.
12     begin(), s2.end(),
13     std::inserter(s, s.begin()));
14     return s;
15 }
16
17 template <class T, class CMP = std::less<T>, class
  ALLOC = std::allocator<T> >
18 std::set<T, CMP, ALLOC> operator + (
19     const std::set<T, CMP, ALLOC> &s1, const std::set<T
20     , CMP, ALLOC> &s2)
21 {
22     std::set<T, CMP, ALLOC> s;

```

```

20 std::set_union(s1.begin(), s1.end(), s2.begin(), s2
    .end(),
21 std::inserter(s, s.begin()));
22 return s;
23 }

```

8 String

8.1 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 }

```

8.2 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)
7     , P(P), n(s.size()), h(n), hi(n), p(n) {
8         for (int i=0; i<n; i++) p[i] = (i ? P*p[i-1]:1)
9         % MOD;
10        for (int i=0; i<n; i++)
11            h[i] = (s[i] + (i ? h[i-1]:0) * P) % MOD;
12        for (int i=n-1; i>=0; i--)
13            hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * P)
14            % MOD;
15    }
16    int query(int l, int r) {
17        ll hash = (h[r] - (l ? h[l-1]*p[r-l+1]:MOD :
18        0));
19        return hash < 0 ? hash + MOD : hash;
20    }
21    int query_inv(int l, int r) {
22        ll hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-l
23        +1] % MOD : 0));
24        return hash < 0 ? hash + MOD : hash;
25    }
26 };
27
28 struct DoubleHash {
29     const ll MOD1 = 90264469;
30     const ll MOD2 = 25699183;
31
32     Hash hash1, hash2;
33
34     DoubleHash();
35
36     DoubleHash(string s) : hash1(s, MOD1), hash2(s,
37     MOD2) {}

```

```

38 pair<int, int> query(int l, int r) {
39     return { hash1.query(l, r), hash2.query(l, r)
40 };
41 }
42
43 pair<int, int> query_inv(int l, int r) {
44     return { hash1.query_inv(l, r), hash2.
45     query_inv(l, r) };
46 }
47
48 };
49
50 struct TripleHash {
51     const ll MOD1 = 90264469;
52     const ll MOD2 = 25699183;
53     const ll MOD3 = 81249169;
54
55     Hash hash1, hash2, hash3;
56
57     TripleHash();
58
59     TripleHash(string s) : hash1(s, MOD1), hash2(s,
60     MOD2), hash3(s, MOD3) {}
61
62     tuple<int, int, int> query(int l, int r) {
63         return { hash1.query(l, r), hash2.query(l, r)
64         , hash3.query(l, r) };
65     }
66
67     tuple<int, int, int> query_inv(int l, int r) {
68         return { hash1.query_inv(l, r), hash2.
69         query_inv(l, r), hash3.query_inv(l, r) };
70     }
71 };
72
73 struct HashK {
74     vector<ll> primes; // more primes = more hashes
75     vector<Hash> hash;
76
77     HashK();
78
79     HashK(string s, vector<ll> primes): primes(primes)
80     {
81         for (auto p : primes) {
82             hash.push_back(Hash(s, p));
83         }
84     }
85
86     vector<int> query(int l, int r) {
87         vector<int> ans;
88
89         for (auto h : hash) {
90             ans.push_back(h.query(l, r));
91         }
92
93         return ans;
94     }
95
96     vector<int> query_inv(int l, int r) {
97         vector<int> ans;
98
99         for (auto h : hash) {
100             ans.push_back(h.query_inv(l, r));
101         }
102
103         return ans;
104     }
105 };

```

8.3 Is Substring

```

1 // equivalente ao in do python
2

```

```

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

```

8.4 Trie Xor

```

1 // TrieXOR
2 //
3 // adiciona, remove e verifica se existe strings
4 // binarias
5 // max_xor(x) = maximiza o xor de x com algum valor
6 // da trie
7 //
8 // https://codeforces.com/problemset/problem/706/D
9 //
10 // O(|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 & (1 << i)) > 0);
32
33             if (trie[curr][b] == 0)
34                 trie[curr][b] = nxt++;
35
36             curr = trie[curr][b];
37         }
38
39         finish[curr]++;
40     }
41
42     void rem(int x) {
43         int curr = 0;
44
45         for (int i = 31; i >= 0; i--) {
46             int b = ((x & (1 << i)) > 0);
47
48             paths[curr]--;
49             curr = trie[curr][b];
50         }
51
52         paths[curr]--;
53         finish[curr]--;
54     }
55
56     int search(int x) {
57         int curr = 0;
58
59         for (int i = 31; i >= 0; i--) {
60             int b = ((x & (1 << i)) > 0);
61
62             if (trie[curr][b] == 0) return false;
63
64             curr = trie[curr][b];
65         }
66
67         return (finish[curr] > 0);
68     }
69
70     int max_xor(int x) { // maximum xor with x and
71 any number of trie
72         int curr = 0, ans = 0;
73
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[
79 curr][want]] == 0) want ^= 1;
80             if (trie[curr][want] == 0 || paths[trie[
81 curr][want]] == 0) break;
82             if (want != b) ans |= (1 << i);
83
84             curr = trie[curr][want];
85         }
86
87         return ans;
88     }
89 };

```

```

34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87

```