

Contents

1 First Thing First

1.1	Header	1
1.2	55kai	1

2 Data Structure

2.1	RMQ	2
2.2	Segment Tree Beats	2
2.3	Segment Tree	2
2.4	K-D Tree	3
2.5	STL+	3
2.6	BIT	3
2.7	Trie	4
2.8	Treap	4
2.9	Cartesian Tree	6
2.10	LCT	6
2.11	Mo's Algorithm On Tree	7
2.12	CDQ's Divide and Conquer	8
2.13	Persistent Segment Tree	8
2.14	Persistent Union Find	8

3 Math

3.1	Multiplication, Powers	9
3.2	Matrix Power	9
3.3	Sieve	9
3.4	Prime Test	10
3.5	Pollard-Rho	10
3.6	Berlekamp-Massey	10
3.7	Extended Euclidean	11
3.8	Inverse	11
3.9	Binomial Numbers	11
3.10	NTT, FFT, FWT	11
3.11	Simpson's Numerical Integration	12
3.12	Gauss Elimination	12
3.13	Factor Decomposition	12
3.14	Primitive Root	12
3.15	Quadratic Residue	12
3.16	Chinese Remainder Theorem	13
3.17	Bernoulli Numbers	13
3.18	Simplex Method	13
3.19	BSGS	13

4 Graph Theory

4.1	LCA	14
4.2	Maximum Flow	14
4.3	Minimum Cost Maximum Flow	14
4.4	Path Intersection on Trees	14
4.5	Centroid Decomposition (Divide-Conquer)	14
4.6	Heavy-light Decomposition	15
4.7	Bipartite Matching	16
4.8	Virtual Tree	16
4.9	Euler Tour	16
4.10	SCC, 2-SAT	17
4.11	Topological Sort	17
4.12	General Matching	17
4.13	Tarjan	17
4.14	Bi-connected Components, Block-cut Tree	18
4.15	Minimum Directed Spanning Tree	18
4.16	Cycles	19
4.17	Dominator Tree	19
4.18	Global Minimum Cut	19

5 Geometry

5.1	2D Basics	20
5.2	Polar angle sort	20
5.3	Segments, lines	20
5.4	Polygons	20
5.5	Half-plane intersection	21

5.6	Circles	21
5.7	Circle Union	22
5.8	Minimum Covering Circle	23
5.9	Circle Inversion	23
5.10	3D Basics	23
5.11	3D Line, Face	23
5.12	3D Convex	24

6 String

6.1	Aho-Corasick Automation	24
6.2	Hash	24
6.3	KMP	25
6.4	Manacher	25
6.5	Palindrome Automation	25
6.6	Suffix Array	25
6.7	Suffix Automation	27

7 Miscellaneous

7.1	Date	28
7.2	Subset Enumeration	28
7.3	Digit DP	28
7.4	Simulated Annealing	28

1 First Thing First

1.1 Header

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using LL = long long;
4 #define FOR(i, x, y) for (decay<decltype(y)> i = (x), _##i = (y); i < _##i; ++i)
5 #define FORD(i, x, y) for (decay<decltype(x)> i = (x), _##i = (y); i > _##i; --i)
6 #ifndef zerol
7 #define dbg(x...) do { cout << "\033[32;1m" << #x << " -> "; err(x); } while (0)
8 void err() { cout << "\033[39;0m" << endl; }
9 template<template<typename...> class T, typename t, typename... A>
10 void err(T<t> a, A... x) { for (auto v: a) cout << v << ' '; err(x...); }
11 template<typename T, typename... A>
12 void err(T a, A... x) { cout << a << ' '; err(x...); }
13 #else
14 #define dbg(...)
15 #endif

```

1.2 55kai

```

1 inline char nc() {
2     static char buf[100000], *p1 = buf, *p2 = buf;
3     return p1 == p2 && (p2 = (p1 = buf) + fread(buf, 1, 100000, stdin), p1 == p2) ? EOF : *p1++;
4 }
5 template<typename T>
6 bool rn(T& v) {
7     static char ch;
8     while (ch != EOF && !isdigit(ch)) ch = nc();
9     if (ch == EOF) return false;

```

```

10     for (v = 0; isdigit(ch); ch = nc())
11         v = v * 10 + ch - '0';
12     return true;
13 }
14
15 template<typename T>
16 void o(T p) {
17     static int stk[70], tp;
18     if (p == 0) { putchar('0'); return; }
19     if (p < 0) { p = -p; putchar('-'); }
20     while (p) stk[++tp] = p % 10, p /= 10;
21     while (tp) putchar(stk[tp--] + '0');
22 }

```

2 Data Structure

2.1 RMQ

```

1 int f[maxn][maxn][10][10];
2 inline int highbit(int x) { return 31 - __builtin_clz(x); }
3 inline int calc(int x, int y, int xx, int yy, int p, int q) {
4     return max(
5         max(f[x][y][p][q], f[xx - (1 << p) + 1][yy - (1 << q) + 1][p][q]),
6         max(f[xx - (1 << p) + 1][y][p][q], f[x][yy - (1 << q) + 1][p][q]));
7 }
8
9 void init() {
10     FOR(x, 0, highbit(n) + 1)
11     FOR(y, 0, highbit(m) + 1)
12     FOR(i, 0, n - (1 << x) + 1)
13     FOR(j, 0, m - (1 << y) + 1) {
14         if (!x && !y) { f[i][j][x][y] = a[i][j]; continue; }
15         f[i][j][x][y] = calc(i, j, i + (1 << x) - 1, j + (1 << y) - 1, max(x - 1, 0), max(y - 1, 0));
16     }
17 }
18
19 inline int get_max(int x, int y, int xx, int yy) {
20     return calc(x, y, xx, yy, highbit(xx - x + 1), highbit(yy - y + 1));
21 }
22
23 struct RMQ {
24     int f[22][M];
25     inline int highbit(int x) { return 31 - __builtin_clz(x); }
26     void init(int* v, int n) {
27         FOR(i, 0, n) f[0][i] = v[i];
28         FOR(x, 1, highbit(n) + 1)
29         FOR(i, 0, n - (1 << x) + 1)
30             f[x][i] = min(f[x - 1][i], f[x - 1][i + (1 << (x - 1))]);
31     }
32 }

```

```

35 int get_min(int l, int r) {
36     assert(l <= r);
37     int t = highbit(r - l + 1);
38     return min(f[t][l], f[t][r - (1 << t)
39               + 1]);
40 } rmq;

```

2.2 Segment Tree Beats

```

1 namespace R {
2 #define lson o * 2, l, (l + r) / 2
3 #define rson o * 2 + 1, (l + r) / 2 + 1, r
4 int ml[N], m2[N], cml[N];
5 LL sum[N];
6 void up(int o) {
7     int lc = o * 2, rc = lc + 1;
8     ml[o] = max(ml[lc], ml[rc]);
9     sum[o] = sum[lc] + sum[rc];
10    if (ml[lc] == ml[rc]) {
11        cml[o] = cml[lc] + cml[rc];
12        m2[o] = max(m2[lc], m2[rc]);
13    } else {
14        cml[o] = ml[lc] > ml[rc] ? cml[lc]
15              : cml[rc];
16        m2[o] = max(min(ml[lc], ml[rc]),
17                  max(m2[lc], m2[rc]));
18    }
19 }
20 void mod(int o, int x) {
21     if (x >= ml[o]) return;
22     assert(x > m2[o]);
23     sum[o] -= 1LL * (ml[o] - x) * cml[o];
24     ml[o] = x;
25 }
26 void down(int o) {
27     int lc = o * 2, rc = lc + 1;
28     mod(lc, ml[o]); mod(rc, ml[o]);
29 }
30 void build(int o, int l, int r) {
31     if (l == r) { int t; read(t); sum[o] = ml[o] = t; m2[o] = -INF; cml[o] = 1; }
32     else { build(lson); build(rson); up(o); }
33 }
34 void update(int ql, int qr, int x, int o, int l, int r) {
35     if (r < ql || qr < l || ml[o] <= x) return;
36     if (ql <= l && r <= qr && m2[o] < x) { mod(o, x); return; }
37     down(o);
38     update(ql, qr, x, lson); update(ql, qr, x, rson);
39     up(o);
40 }
41 int qmax(int ql, int qr, int o, int l, int r) {
42     if (r < ql || qr < l) return -INF;
43     if (ql <= l && r <= qr) return ml[o];
44     down(o);

```

```

43     return max(qmax(ql, qr, lson), qmax(ql, qr, rson));
44 }
45 LL qsum(int ql, int qr, int o, int l, int r) {
46     if (r < ql || qr < l) return 0;
47     if (ql <= l && r <= qr) return sum[o];
48     down(o);
49     return qsum(ql, qr, lson) + qsum(ql, qr, rson);
50 }
51 }

```

2.3 Segment Tree

```

1 // set + add
2
3 struct IntervalTree {
4 #define ls o * 2, l, m
5 #define rs o * 2 + 1, m + 1, r
6     static const LL M = maxn * 4, RS = 1E18 - 1;
7     LL addv[M], setv[M], minv[M], maxv[M], sumv[M];
8     void init() {
9         memset(addv, 0, sizeof addv);
10        fill(setv, setv + M, RS);
11        memset(minv, 0, sizeof minv);
12        memset(maxv, 0, sizeof maxv);
13        memset(sumv, 0, sizeof sumv);
14    }
15    void maintain(LL o, LL l, LL r) {
16        if (l < r) {
17            LL lc = o * 2, rc = o * 2 + 1;
18            sumv[o] = sumv[lc] + sumv[rc];
19            minv[o] = min(minv[lc], minv[rc]);
20            maxv[o] = max(maxv[lc], maxv[rc]);
21        } else sumv[o] = minv[o] = maxv[o] = 0;
22        if (setv[o] != RS) { minv[o] = maxv[o] = setv[o]; sumv[o] = setv[o] * (r - l + 1); }
23        if (addv[o] != 0) { minv[o] += addv[o]; maxv[o] += addv[o]; sumv[o] += addv[o] * (r - l + 1); }
24    }
25    void build(LL o, LL l, LL r) {
26        if (l == r) addv[o] = a[l];
27        else {
28            LL m = (l + r) / 2;
29            build(ls); build(rs);
30        }
31        maintain(o, l, r);
32    }
33    void pushdown(LL o) {
34        LL lc = o * 2, rc = o * 2 + 1;
35        if (setv[o] != RS) { setv[lc] = setv[rc] = setv[o]; addv[lc] = addv[rc] = 0; setv[o] = RS; }
36    }

```

```

40     if (addv[o]) {
41         addv[lc] += addv[o]; addv[rc] += addv[o];
42         addv[o] = 0;
43     }
44 }
45 void update(LL p, LL q, LL o, LL l, LL r, LL v, LL op) {
46     if (p <= r && l <= q) {
47         if (p <= l && r <= q) {
48             if (op == 2) { setv[o] = v; addv[o] = 0; }
49             else addv[o] += v;
50         } else {
51             pushdown(o);
52             LL m = (l + r) / 2;
53             update(p, q, ls, v, op); update(p, q, rs, v, op);
54         }
55     }
56     maintain(o, l, r);
57 }
58 void query(LL p, LL q, LL o, LL l, LL r, LL add, LL& ssum, LL& smin, LL& smax) {
59     if (p > r || l > q) return;
60     if (setv[o] != RS) {
61         LL v = setv[o] + add + addv[o];
62         ssum += v * (min(r, q) - max(l, p) + 1);
63         smin = min(smin, v);
64         smax = max(smax, v);
65     } else if (p <= l && r <= q) {
66         ssum += sumv[o] + add * (r - l + 1);
67         smin = min(smin, minv[o] + add);
68         smax = max(smax, maxv[o] + add);
69     } else {
70         LL m = (l + r) / 2;
71         query(p, q, ls, add + addv[o], ssum, smin, smax);
72         query(p, q, rs, add + addv[o], ssum, smin, smax);
73     }
74 } IT;
75
76 // persistent
77 namespace tree {
78 #define mid ((l + r) >> 1)
79 #define lson ql, qr, l, mid
80 #define rson ql, qr, mid + 1, r
81 struct P {
82     LL add, sum;
83     int ls, rs;
84 } tr[maxn * 45 * 2];
85 int sz = 1;
86 int N(LL add, int l, int r, int ls, int rs) {
87     tr[sz] = {add, tr[ls].sum + tr[rs].sum + add * (len[r] - len[l - 1])};
88     return sz++;
89 }
90 }
91

```

```

92 int update(int o, int ql, int qr, int l,
93           int r, LL add) {
94     if (ql > r || l > qr) return o;
95     const P& t = tr[o];
96     if (ql <= l && r <= qr) return N(add
97         + t.add, l, r, t.ls, t.rs);
98     return N(t.add, l, r, update(t.ls,
99         lson, add), update(t.rs, rson,
100         add));
101 }
102 LL query(int o, int ql, int qr, int l,
103          int r, LL add = 0) {
104     if (ql > r || l > qr) return 0;
105     const P& t = tr[o];
106     if (ql <= l && r <= qr) return add *
107         (len[r] - len[l - 1]) + t.sum;
108     return query(t.ls, lson, add + t.add)
109         + query(t.rs, rson, add + t.add);
110 }

```

2.4 K-D Tree

```

1 // global variable pruning
2 // visit L/R with more potential
3 namespace kd {
4     const int K = 2, inf = 1E9, M = N;
5     const double lim = 0.7;
6     struct P {
7         int d[K], l[K], r[K], sz, val;
8         LL sum;
9         P *ls, *rs;
10        P* up() {
11            sz = ls->sz + rs->sz + 1;
12            sum = ls->sum + rs->sum + val;
13            FOR (i, 0, K) {
14                l[i] = min(d[i], min(ls->l[i],
15                    rs->l[i]));
16                r[i] = max(d[i], max(ls->r[i],
17                    rs->r[i]));
18            }
19            return this;
20        }
21    } pool[M], *null = new P, *pit = pool;
22    static P *tmp[M], **pt;
23    void init() {
24        null->ls = null->rs = null;
25        FOR (i, 0, K) null->l[i] = inf, null
26            ->r[i] = -inf;
27        null->sum = null->val = 0;
28        null->sz = 0;
29    }
30    P* build(P** l, P** r, int d = 0) { // [l
31        , r)
32        if (d == K) d = 0;
33        if (l == r) return null;
34        P** m = l + (r - l) / 2; assert(l <=
35            m && m < r);
36        nth_element(l, m, r, [&](const P* a,
37            const P* b) {
38            return a->d[d] < b->d[d];
39        });

```

```

35 P* o = *m;
36 o->ls = build(l, m, d + 1); o->rs =
37     build(m + 1, r, d + 1);
38     return o->up();
39 }
40 P* Build() {
41     pt = tmp; FOR (it, pool, pit) *pt++ =
42         it;
43     return build(tmp, pt);
44 }
45 inline bool inside(int p[], int q[], int
46     l[], int r[]) {
47     FOR (i, 0, K) if (r[i] < q[i] || p[i]
48         < l[i]) return false;
49     return true;
50 }
51 LL query(P* o, int l[], int r[]) {
52     if (o == null) return 0;
53     FOR (i, 0, K) if (o->r[i] < l[i] || r
54         [i] < o->l[i]) return 0;
55     if (inside(o->l, o->r, l, r)) return
56         o->sum;
57     return query(o->ls, l, r) + query(o->
58         rs, l, r) +
59         (inside(o->d, o->d, l, r) ? o
60             ->val : 0);
61 }
62 void dfs(P* o) {
63     if (o == null) return;
64     *pt++ = o; dfs(o->ls); dfs(o->rs);
65 }
66 P* ins(P* o, P* x, int d = 0) {
67     if (d == K) d = 0;
68     if (o == null) return x->up();
69     P& oo = x->d[d] <= o->d[d] ? o->ls :
70         o->rs;
71     if (oo->sz > o->sz * lim) {
72         pt = tmp; dfs(o); *pt++ = x;
73         return build(tmp, pt, d);
74     }
75     oo = ins(oo, x, d + 1);
76     return o->up();
77 }

```

2.5 STL+

```

1 // priority_queue
2
3 // binary_heap_tag
4 // pairing_heap_tag: support editing
5 // thin_heap_tag: fast when increasing, can't
6     join
7 #include <ext/pb_ds/priority_queue.hpp>
8 using namespace __gnu_pbds;
9
10 typedef __gnu_pbds::priority_queue<LL, less<
11     LL>, pairing_heap_tag> PQ;
12 __gnu_pbds::priority_queue<int, cmp,
13     pairing_heap_tag>::point_iterator it;
14 PQ pq, pq2;
15
16 int main() {
17     auto it = pq.push(2);

```

```

15 pq.push(3);
16 assert(pq.top() == 3);
17 pq.modify(it, 4);
18 assert(pq.top() == 4);
19 pq2.push(5);
20 pq.join(pq2);
21 assert(pq.top() == 5);
22 }
23
24 // BBT
25
26 // ov_tree_tag
27 // rb_tree_tag
28 // splay_tree_tag
29
30 // mapped: null_type or null_mapped_type (
31     old) is null
32 // Node_Update should be
33     tree_order_statistics_node_update to use
34     find_by_order & order_of_key
35 // find_by_order: find the element with order
36     +1 (0-based)
37 // order_of_key: number of elements lt r_key
38 // support join & split
39
40 #include <ext/pb_ds/assoc_container.hpp>
41 using namespace __gnu_pbds;
42 using Tree = tree<int, null_type, less<int>,
43     rb_tree_tag,
44     tree_order_statistics_node_update>;
45 Tree t;
46
47 // Persistent BBT
48
49 #include <ext/rope>
50 using namespace __gnu_cxx;
51 rope<int> s;
52
53 int main() {
54     FOR (i, 0, 5) s.push_back(i); // 0 1 2 3
55     4
56     s.replace(1, 2, s); // 0 (0 1 2 3 4) 3 4
57     auto ss = s.substr(2, 2); // 1 2
58     s.erase(2, 2); // 0 1 4
59     s.insert(2, s); // equal to s.replace(2,
60         0, s)
61     assert(s[2] == s.at(2)); // 2
62 }
63
64 // Hash Table
65
66 #include <ext/pb_ds/assoc_container.hpp>
67 #include <ext/pb_ds/hash_policy.hpp>
68 using namespace __gnu_pbds;
69
70 gp_hash_table<int, int> mp;
71 cc_hash_table<int, int> mp;

```

2.6 BIT

```

1 namespace bit {
2     LL c[M];
3     inline int lowbit(int x) { return x & -x;
4 }

```

```

4 void add(int x, LL v) {
5     for (; x < M; x += lowbit(x))
6         c[x] += v;
7 }
8 LL sum(int x) {
9     LL ret = 0;
10    for (; x > 0; x -= lowbit(x))
11        ret += c[x];
12    return ret;
13 }
14 int kth(LL k) {
15     int p = 0;
16     for (int lim = 1 << 20; lim; lim /= 2)
17         if (p + lim < M && c[p + lim] < k)
18             p += lim;
19     return p + 1;
20 }
21 }
22 }
23 namespace bit {
24     int c[maxn], cc[maxn];
25     inline int lowbit(int x) { return x & -x; }
26 }
27 void add(int x, int v) {
28     for (int i = x; i <= n; i += lowbit(i)) {
29         c[i] += v; cc[i] += x * v;
30     }
31 }
32 void add(int l, int r, int v) { add(l, v);
33     add(r + 1, -v); }
34 int sum(int x) {
35     int ret = 0;
36     for (int i = x; i > 0; i -= lowbit(i))
37         ret += (x + 1) * c[i] - cc[i];
38     return ret;
39 }
40 int sum(int l, int r) { return sum(r) -
41     sum(l - 1); }
42 }
43 namespace bit {
44     LL c[N], cc[N], ccc[N];
45     inline LL lowbit(LL x) { return x & -x; }
46     void add(LL x, LL v) {
47         for (LL i = x; i < N; i += lowbit(i)) {
48             c[i] = (c[i] + v) % MOD;
49             cc[i] = (cc[i] + x * v) % MOD;
50             ccc[i] = (ccc[i] + x * x % MOD *
51                 v) % MOD;
52         }
53     }
54     void add(LL l, LL r, LL v) { add(l, v);
55         add(r + 1, -v); }
56     LL sum(LL x) {
57         static LL INV2 = (MOD + 1) / 2;
58         LL ret = 0;
59         for (LL i = x; i > 0; i -= lowbit(i))
60             ret += (x + 1) * (x + 2) % MOD *
61                 c[i] % MOD
62                 - (2 * x + 3) * cc[i] %
63                 MOD

```

2.7 Trie

```

1 namespace trie {
2     const int M = 31;
3     int ch[N * M][2], sz;
4     void init() { memset(ch, 0, sizeof ch);
5         sz = 2; }
6     void ins(LL x) {
7         int u = 1;
8         FOR (i, M, -1) {
9             bool b = x & (1LL << i);
10            if (!ch[u][b]) ch[u][b] = sz++;
11            u = ch[u][b];
12        }
13    }
14 }
15 // persistent
16 // !!! sz = 1
17 struct P { int w, ls, rs; };
18 P tr[M] = {{0, 0, 0}};
19 int sz;
20 int _new(int w, int ls, int rs) { tr[sz] = {w,
21     ls, rs}; return sz++; }
22 int ins(int oo, int v, int d = 30) {
23     P& o = tr[oo];
24     if (d == -1) return _new(o.w + 1, 0, 0);
25     bool u = v & (1 << d);
26     return _new(o.w + 1, u = 0 ? ins(o.ls, v,
27         d - 1) : o.ls, u = 1 ? ins(o.rs, v,
28         d - 1) : o.rs);
29 }
30 int query(int pp, int qq, int v, int d = 30)
31 {
32     if (d == -1) return 0;
33     bool u = v & (1 << d);
34     P& p = tr[pp], &q = tr[qq];
35     int lw = tr[q.ls].w - tr[p.ls].w;
36     int rw = tr[q.rs].w - tr[p.rs].w;
37     int ret = 0;
38     if (u == 0) {
39         if (rw) { ret += 1 << d; ret += query
40             (p.rs, q.rs, v, d - 1); }
41         else ret += query(p.ls, q.ls, v, d -
42             1);
43     } else {
44         if (lw) { ret += 1 << d; ret += query
45             (p.ls, q.ls, v, d - 1); }
46         else ret += query(p.rs, q.rs, v, d -
47             1);
48     }
49     return ret;
50 }

```

2.8 Treap

```

1 // set
2 namespace treap {
3     const int M = maxn * 17;
4     extern struct P* const null;
5     struct P {
6         P* ls, *rs;
7         int v, sz;
8         unsigned rd;
9         P(int v): ls(null), rs(null), v(v),
10             sz(1), rd(rnd()) {}
11         P(): sz(0) {}
12     }
13     P* up() { sz = ls->sz + rs->sz + 1;
14         return this; }
15     int lower(int v) {
16         if (this == null) return 0;
17         return this->v >= v ? ls->lower(v)
18             : rs->lower(v) + ls->sz +
19             1;
20     }
21     int upper(int v) {
22         if (this == null) return 0;
23         return this->v > v ? ls->upper(v)
24             : rs->upper(v) + ls->sz + 1;
25     }
26 } *const null = new P, pool[M], *pit =
27     pool;
28 P* merge(P* l, P* r) {
29     if (l == null) return r; if (r ==
30     null) return l;
31     if (l->rd < r->rd) { l->rs = merge(l
32     ->rs, r); return l->up(); }
33     else { r->ls = merge(l, r->ls);
34         return r->up(); }
35 }
36 void split(P* o, int rk, P*& l, P*& r) {
37     if (o == null) { l = r = null; return
38     ; }
39     if (o->ls->sz >= rk) { split(o->ls,
40     rk, l, o->ls); r = o->up(); }
41     else { split(o->rs, rk - o->ls->sz -
42     1, o->rs, r); l = o->up(); }
43 }
44 }
45 // persistent set
46 namespace treap {
47     const int M = maxn * 17 * 12;
48     extern struct P* const null, *pit;
49     struct P {
50         P* ls, *rs;
51         int v, sz;
52         LL sum;
53         P(P* ls, P* rs, int v): ls(ls), rs(rs),
54             v(v), sz(ls->sz + rs->sz + 1),
55             sum(

```

sum

```

45 P() {}
46
47 void* operator new(size_t _) { return
48     pit++; }
49 template<typename T>
50 int rk(int v, T&& cmp) {
51     if (this == null) return 0;
52     return cmp(this->v, v) ? ls->rk(v,
53         cmp) : rs->rk(v, cmp) + ls
54         ->sz + 1;
55 }
56 int lower(int v) { return rk(v,
57     greater_equal<int>()); }
58 int upper(int v) { return rk(v,
59     greater<int>()); }
60 } pool[M], *pit = pool, *const null = new
61 P;
62 P* merge(P* l, P* r) {
63     if (l == null) return r; if (r ==
64     null) return l;
65     if (rnd() % (l->sz + r->sz) < l->sz)
66         return new P{l->ls, merge(l->rs,
67             r), l->v};
68     else return new P{merge(l, r->ls), r
69         ->rs, r->v};
70 }
71 void split(P* o, int rk, P*& l, P*& r) {
72     if (o == null) { l = r = null; return
73     ; }
74     if (o->ls->sz >= rk) { split(o->ls,
75         rk, l, r); r = new P{r, o->rs, o
76         ->v}; }
77     else { split(o->rs, rk - o->ls->sz -
78         1, l, r); l = new P{o->ls, l, o->
79         v}; }
80 }
81 // persistent set with pushdown
82 int now;
83 namespace Treap {
84     const int M = 10000000;
85     extern struct P* const null, *pit;
86     struct P {
87         P* ls, *rs;
88         int sz, time;
89         LL cnt, sc, pos, add;
90         bool rev;
91
92         P* up() { sz = ls->sz + rs->sz + 1;
93             sc = ls->sc + rs->sc + cnt;
94             return this; } // MOD
95         P* check() {
96             if (time == now) return this;

```

```

81 P* t = new(pit++) P; *t = *this;
82     t->time = now; return t;
83 };
84 P* _do_rev() { rev ^= 1; add *= -1;
85     pos *= -1; swap(ls, rs); return
86     this; } // MOD
87 P* _do_add(LL v) { add += v; pos += v
88     ; return this; } // MOD
89 P* do_rev() { if (this == null)
90     return this; return check()->
91     _do_rev(); } // FIX & MOD
92 P* do_add(LL v) { if (this == null)
93     return this; return check()->
94     _do_add(v); } // FIX & MOD
95 P* _down() { // MOD
96     if (rev) { ls = ls->do_rev(); rs
97         = rs->do_rev(); rev = 0; }
98     if (add) { ls = ls->do_add(add);
99         rs = rs->do_add(add); add =
100         0; }
101     return this;
102 }
103 P* down() { return check()->_down();
104 } // FIX & MOD
105 void _split(LL p, P*& l, P*& r) { //
106     MOD
107     if (pos >= p) { ls->split(p, l, r
108         ); ls = r; r = up(); }
109     else { rs->split(p, l, r
110         ); rs = l; l = up(); }
111 }
112 void split(LL p, P*& l, P*& r) { //
113     FIX & MOD
114     if (this == null) l = r = null;
115     else down()->_split(p, l, r);
116 }
117 } pool[M], *pit = pool, *const null = new
118 P;
119 P* merge(P* a, P* b) {
120     if (a == null) return b; if (b ==
121     null) return a;
122     if (rand() % (a->sz + b->sz) < a->sz)
123         { a = a->down(); a->rs = merge(a
124             ->rs, b); return a->up(); }
125     else
126         { b = b->down(); b->ls = merge(a,
127             b->ls); return b->up(); }
128 }
129 // sequence with add, sum
130 namespace treap {
131     const int M = 8E5 + 100;
132     extern struct P* const null;
133     struct P {
134         P* ls, *rs;
135         int sz, val, add, sum;
136         P(int v, P* ls = null, P* rs = null):
137             ls(ls), rs(rs), sz(1), val(v),
138             add(0), sum(v) {}
139         P(): sz(0), val(0), add(0), sum(0) {}
140
141         P* up() {
142             assert(this != null);
143             sz = ls->sz + rs->sz + 1;
144             sum = ls->sum + rs->sum + val +
145                 add * sz;

```

```

122     return this;
123 }
124 void upd(int v) {
125     if (this == null) return;
126     add += v;
127     sum += sz * v;
128 }
129 P* down() {
130     if (add) {
131         ls->upd(add); rs->upd(add);
132         val += add;
133         add = 0;
134     }
135     return this;
136 }
137
138 P* select(int rk) {
139     if (rk == ls->sz + 1) return this
140     ;
141     return ls->sz >= rk ? ls->select(
142         rk) : rs->select(rk - ls->sz
143         - 1);
144 }
145 pool[M], *pit = pool, *const null = new
146 P, *rt = null;
147
148 P* merge(P* a, P* b) {
149     if (a == null) return b->up();
150     if (b == null) return a->up();
151     if (rand() % (a->sz + b->sz) < a->sz)
152         {
153             a->down()->rs = merge(a->rs, b);
154             return a->up();
155         }
156     else {
157         b->down()->ls = merge(a, b->ls);
158         return b->up();
159     }
160 }
161
162 void split(P* o, int rk, P*& l, P*& r) {
163     if (o == null) { l = r = null; return
164     ; }
165     o->down();
166     if (o->ls->sz >= rk) {
167         split(o->ls, rk, l, o->ls);
168         r = o->up();
169     }
170     else {
171         split(o->rs, rk - o->ls->sz - 1,
172             o->rs, r);
173         l = o->up();
174     }
175 }
176
177 inline void insert(int k, int v) {
178     P* l, *r;
179     split(rt, k - 1, l, r);
180     rt = merge(merge(l, new (pit++) P(v)),
181         r);
182 }
183
184 inline void erase(int k) {
185     P* l, *r, *_;
186     split(rt, k - 1, l, t);
187     split(t, 1, _, r);
188     rt = merge(l, r);
189 }

```

2.9 Cartesian Tree

```

1 void build() {
2     static int s[N], last;
3     int p = 0;
4     FOR (x, 1, n + 1) {
5         last = 0;
6         while (p && val[s[p - 1]] > val[x])
7             last = s[--p];
8         if (p) G[s[p - 1]][1] = x;
9         if (last) G[last][0] = x;
10        s[p++] = x;
11    }
12    rt = s[0];
13 }

```

2.10 LCT

```

1 // do not forget down when findint L/R most
2 // make_root if not sure
3
4 namespace lct {
5     extern struct P *const null;
6     const int M = N;
7     struct P {
8         P *fa, *ls, *rs;
9         int v, maxv;
10        bool rev;
11
12        bool has_fa() { return fa->ls == this
13            || fa->rs == this; }
14        bool d() { return fa->ls == this; }
15        P* c(bool x) { return x ? ls : rs; }
16        void do_rev() {
17            if (this == null) return;
18            rev ^= 1;
19            swap(ls, rs);
20        }
21        P* up() {
22            maxv = max(v, max(ls->maxv, rs->maxv));
23            return this;
24        }
25        void down() {
26            if (rev) {
27                rev = 0;
28                ls->do_rev(); rs->do_rev();
29            }
30        }
31        void all_down() { if (has_fa()) fa->all_down(); down(); }
32    } *const null = new P{0, 0, 0, 0, 0, 0},
33    pool[M], *pit = pool;
34
35    void rot(P* o) {
36        bool dd = o->d();
37        P *f = o->fa, *t = o->c(!dd);
38        if (f->has_fa()) f->fa->c(f->d()) = o;
39        ; o->fa = f->fa;
40        if (t != null) t->fa = f; f->c(dd) = t;
41        o->c(!dd) = f->up(); f->fa = o;
42    }
43 }

```

```

40 void splay(P* o) {
41     o->all_down();
42     while (o->has_fa()) {
43         if (o->fa->has_fa())
44             rot(o->d()) ^ o->fa->d() ? o :
45             o->fa;
46         rot(o);
47     }
48     o->up();
49 }
50 void access(P* u, P* v = null) {
51     if (u == null) return;
52     splay(u); u->rs = v;
53     access(u->up()->fa, u);
54 }
55 void make_root(P* o) {
56     access(o); splay(o); o->do_rev();
57 }
58 void split(P* o, P* u) {
59     make_root(o); access(u); splay(u);
60 }
61 void link(P* u, P* v) {
62     make_root(u); u->fa = v;
63 }
64 void cut(P* u, P* v) {
65     split(u, v);
66     u->fa = v->ls = null; v->up();
67 }
68 bool adj(P* u, P* v) {
69     split(u, v);
70     return v->ls == u && u->ls == null &&
71     u->rs == null;
72 }
73 bool linked(P* u, P* v) {
74     split(u, v);
75     return u == v || u->fa != null;
76 }
77 P* findrt(P* o) {
78     access(o); splay(o);
79     while (o->ls != null) o = o->ls;
80     return o;
81 }
82 P* findfa(P* rt, P* u) {
83     split(rt, u);
84     u = u->ls;
85     while (u->rs != null) {
86         u = u->rs;
87         u->down();
88     }
89     return u;
90 }
91 // maintain subtree size
92 P* up() {
93     sz = ls->sz + rs->sz + _sz + 1;
94     return this;
95 }
96 void access(P* u, P* v = null) {
97     if (u == null) return;
98     splay(u);
99     u->_sz += u->rs->sz - v->sz;
100    u->rs = v;
101    access(u->up()->fa, u);
102 }
103 void link(P* u, P* v) {
104     split(u, v);
105 }

```



```

104 u->fa = v; v->_sz += u->sz;
105 v->up();
106 }
107
108 ////////////////
109 // latest spanning tree
110 ////////////////
111 namespace lct {
112     extern struct P* null;
113     struct P {
114         P *fa, *ls, *rs;
115         int v;
116         P *minp;
117         bool rev;
118
119         bool has_fa() { return fa->ls == this
120             || fa->rs == this; }
121         bool d() { return fa->ls == this; }
122         P& c(bool x) { return x ? ls : rs; }
123         void do_rev() { if (this == null)
124             return; rev ^= 1; swap(ls, rs); }
125
126         P* up() {
127             minp = this;
128             if (minp->v > ls->minp->v) minp =
129                 ls->minp;
130             if (minp->v > rs->minp->v) minp =
131                 rs->minp;
132             return this;
133         }
134         void down() { if (rev) { rev = 0; ls
135             ->do_rev(); rs->do_rev(); }}
136         void all_down() { if (has_fa()) fa->
137             all_down(); down(); }
138     } *null = new P{0, 0, 0, INF, 0, 0}, pool
139     [maxn], *pit = pool;
140     void rot(P* o) {
141         bool dd = o->d();
142         P *f = o->fa, *t = o->c(!dd);
143         if (f->has_fa()) f->fa->c(f->d()) = o
144             ; o->fa = f->fa;
145         if (t != null) t->fa = f; f->c(dd) =
146             t;
147         o->c(!dd) = f->up(); f->fa = o;
148     }
149     void splay(P* o) {
150         o->all_down();
151         while (o->has_fa()) {
152             if (o->fa->has_fa()) rot(o->d() ^
153                 o->fa->d() ? o : o->fa);
154             rot(o);
155         }
156         o->up();
157     }
158     void access(P* u, P* v = null) {
159         if (u == null) return;
160         splay(u); u->rs = v;
161         access(u->up()->fa, u);
162     }
163     void make_root(P* o) { access(o); splay(o
164         ); o->do_rev(); }
165     void split(P* u, P* v) { make_root(u);
166         access(v); splay(v); }
167     bool linked(P* u, P* v) { split(u, v);
168         return u == v || u->fa != null; }

```

```

156 void link(P* u, P* v) { make_root(u); u->
157     fa = v; }
158 void cut(P* u, P* v) { split(u, v); u->fa
159     = v->ls = null; v->up(); }
160
161 using namespace lct;
162 int n, m;
163 P *p[maxn];
164 struct Q {
165     int tp, u, v, l, r;
166 };
167 vector<Q> q;
168
169 int main() {
170     null->minp = null;
171     cin >> n >> m;
172     FOR (i, 1, n + 1) p[i] = new (pit++) P{
173         null, null, null, INF, p[i], 0};
174     int clk = 0;
175     map<pair<int, int>, int> mp;
176     FOR (i, 0, m) {
177         int tp, u, v; scanf("%d%d%d", &tp, &u
178             , &v);
179         if (u > v) swap(u, v);
180         if (tp == 0) mp.insert({{u, v}, clk})
181             ;
182         else if (tp == 1) {
183             auto it = mp.find({u, v}); assert
184                 (it != mp.end());
185             q.push_back({1, u, v, it->second,
186                 clk});
187             mp.erase(it);
188         } else q.push_back({0, u, v, clk, clk
189             });
190         ++clk;
191     }
192     for (auto& x: mp) q.push_back({1, x.first
193         .first, x.first.second, x.second, clk
194         });
195     sort(q.begin(), q.end(), [](const Q& a,
196         const Q& b)->bool { return a.l < b.l;
197         });
198     map<P*, int> mp2;
199     FOR (i, 0, q.size()) {
200         Q& cur = q[i];
201         int u = cur.u, v = cur.v;
202         if (cur.tp == 0) {
203             if (!linked(p[u], p[v])) puts("N"
204                 );
205             else puts(p[v]->minp->v >= cur.r
206                 ? "Y" : "N");
207             continue;
208         }
209         if (linked(p[u], p[v])) {
210             P* t = p[v]->minp;
211             if (t->v > cur.r) continue;
212             Q& old = q[mp2[t]];
213             cut(p[old.u], t); cut(p[old.v], t
214                 );
215         }
216         P* t = new (pit++) P {null, null,
217             null, cur.r, t, 0};
218         mp2[t] = i;
219         link(t, p[u]); link(t, p[v]);
220     }

```

2.11 Mo's Algorithm On Tree

```

1 struct Q {
2     int u, v, idx;
3     bool operator < (const Q& b) const {
4         const Q& a = *this;
5         return blk[a.u] < blk[b.u] || (blk[a.
6             u] == blk[b.u] && in[a.v] < in[b.
7                 v]);
8     }
9 };
10 void dfs(int u = 1, int d = 0) {
11     static int S[maxn], sz = 0, blk_cnt = 0,
12         clk = 0;
13     in[u] = clk++;
14     dep[u] = d;
15     int btm = sz;
16     for (int v: G[u]) {
17         if (v == fa[u]) continue;
18         fa[v] = u;
19         dfs(v, d + 1);
20         if (sz - btm >= B) {
21             while (sz > btm) blk[S[--sz]] =
22                 blk_cnt;
23             ++blk_cnt;
24         }
25     }
26     S[sz++] = u;
27     if (u == 1) while (sz) blk[S[--sz]] =
28         blk_cnt - 1;
29 }
30 void flip(int k) {
31     dbg(k);
32     if (vis[k]) {
33         // ...
34     } else {
35         // ...
36     }
37     vis[k] ^= 1;
38 }
39 void go(int& k) {
40     if (bug == -1) {
41         if (vis[k] && !vis[fa[k]]) bug = k;
42         if (!vis[k] && vis[fa[k]]) bug = fa[k
43             ];
44     }
45     flip(k);
46     k = fa[k];
47 }
48 void mv(int a, int b) {
49     bug = -1;
50     if (vis[b]) bug = b;
51     if (dep[a] < dep[b]) swap(a, b);
52     while (dep[a] > dep[b]) go(a);
53     while (a != b) {
54         go(a); go(b);
55     }
56     go(a); go(bug);

```

```

55 }
56
57 for (Q& q: query) {
58     mv(u, q.u); u = q.u;
59     mv(v, q.v); v = q.v;
60     ans[q.idx] = Ans;
61 }

```

2.12 CDQ's Divide and Conquer

```

1  const int maxn = 2E5 + 100;
2  struct P {
3      int x, y;
4      int* f;
5      bool d1, d2;
6  } a[maxn], b[maxn], c[maxn];
7  int f[maxn];
8
9  void go2(int l, int r) {
10     if (l + 1 == r) return;
11     int m = (l + r) >> 1;
12     go2(l, m); go2(m, r);
13     FOR (i, l, m) b[i].d2 = 0;
14     FOR (i, m, r) b[i].d2 = 1;
15     merge(b + l, b + m, b + m, b + r, c + l,
16           [(const P& a, const P& b)->bool {
17               if (a.y != b.y) return a.y < b.y;
18               return a.d2 > b.d2;
19           }]);
20     int mx = -1;
21     FOR (i, l, r) {
22         if (c[i].d1 && c[i].d2) *c[i].f = max
23             (*c[i].f, mx + 1);
24         if (!c[i].d1 && !c[i].d2) mx = max(mx
25             , *c[i].f);
26     }
27     FOR (i, l, r) b[i] = c[i];
28 }
29
30 void gol(int l, int r) { // [l, r)
31     if (l + 1 == r) return;
32     int m = (l + r) >> 1;
33     gol(l, m);
34     FOR (i, l, m) a[i].d1 = 0;
35     FOR (i, m, r) a[i].d1 = 1;
36     copy(a + l, a + r, b + l);
37     sort(b + l, b + r, [(const P& a, const P
38         & b)->bool {
39             if (a.x != b.x) return a.x < b.x;
40             return a.d1 > b.d1;
41         }]);
42     go2(l, r);
43     gol(m, r);
44 }

```

2.13 Persistent Segment Tree

```

1  namespace tree {
2      #define mid ((l + r) >> 1)
3      #define lson l, mid
4      #define rson mid + 1, r

```

```

5      const int MAGIC = M * 30;
6      struct P {
7          int sum, ls, rs;
8      } tr[MAGIC] = {{0, 0, 0}};
9      int sz = 1;
10     int N(int sum, int ls, int rs) {
11         if (sz == MAGIC) assert(0);
12         tr[sz] = {sum, ls, rs};
13         return sz++;
14     }
15     int ins(int o, int x, int v, int l = 1,
16             int r = ls) {
17         if (x < l || x > r) return o;
18         const P& t = tr[o];
19         if (l == r) return N(t.sum + v, 0, 0);
20         return N(t.sum + v, ins(t.ls, x, v,
21                                 lson), ins(t.rs, x, v, rson));
22     }
23     int query(int o, int ql, int qr, int l =
24               1, int r = ls) {
25         if (ql > r || l > qr) return 0;
26         const P& t = tr[o];
27         if (ql <= l && r <= qr) return t.sum;
28         return query(t.ls, ql, qr, lson) +
29             query(t.rs, ql, qr, rson);
30     }
31 }
32 // kth
33 int query(int pp, int qq, int l, int r, int k
34           ) { // [pp, qq]
35     if (l == r) return l;
36     const P &p = tr[pp], &q = tr[qq];
37     int w = tr[q.ls].w - tr[p.ls].w;
38     if (k <= w) return query(p.ls, q.ls, lson
39                             , k);
40     else return query(p.rs, q.rs, rson, k - w
41                     );
42 }
43
44 ////////////////
45 // with bit
46 ////////////////
47
48 typedef vector<int> VI;
49 struct TREE {
50     #define mid ((l + r) >> 1)
51     #define lson l, mid
52     #define rson mid + 1, r
53     struct P {
54         int w, ls, rs;
55     } tr[maxn * 20 * 20];
56     int sz = 1;
57     TREE() { tr[0] = {0, 0, 0}; }
58     int N(int w, int ls, int rs) {
59         tr[sz] = {w, ls, rs};
60         return sz++;
61     }
62     int add(int tt, int l, int r, int x, int
63            d) {
64         if (x < l || r < x) return tt;
65         const P& t = tr[tt];
66         if (l == r) return N(t.w + d, 0, 0);
67         return N(t.w + d, add(t.ls, lson, x,
68                               d), add(t.rs, rson, x, d));
69     }

```

```

61     int ls_sum(const VI& rt) {
62         int ret = 0;
63         FOR (i, 0, rt.size())
64             ret += tr[rt[i]].ls.w;
65         return ret;
66     }
67     inline void ls(VI& rt) { transform(rt.
68         begin(), rt.end(), rt.begin(), [&](
69             int x)->int { return tr[x].ls; }); }
70     inline void rs(VI& rt) { transform(rt.
71         begin(), rt.end(), rt.begin(), [&](
72             int x)->int { return tr[x].rs; }); }
73     int query(VI& p, VI& q, int l, int r, int
74              k) {
75         if (l == r) return l;
76         int w = ls_sum(q) - ls_sum(p);
77         if (k <= w) {
78             ls(p); ls(q);
79             return query(p, q, lson, k);
80         }
81         else {
82             rs(p); rs(q);
83             return query(p, q, rson, k - w);
84         }
85     }
86 } tree;
87 struct BIT {
88     int root[maxn];
89     void init() { memset(root, 0, sizeof root
90         ); }
91     inline int lowbit(int x) { return x & -x;
92     }
93     void update(int p, int x, int d) {
94         for (int i = p; i <= m; i += lowbit(i
95             ))
96             root[i] = tree.add(root[i], 1, m,
97                                 x, d);
98     }
99     int query(int l, int r, int k) {
100         VI p, q;
101         for (int i = l - 1; i > 0; i -=
102             lowbit(i)) p.push_back(root[i]);
103         for (int i = r; i > 0; i -= lowbit(i)
104             ) q.push_back(root[i]);
105         return tree.query(p, q, 1, m, k);
106     }
107 } bit;
108
109 void init() {
110     m = 10000;
111     tree.sz = 1;
112     bit.init();
113     FOR (i, 1, m + 1)
114         bit.update(i, a[i], 1);
115 }

```

2.14 Persistent Union Find

```

1  namespace uf {
2      int fa[maxn], sz[maxn];
3      int undo[maxn], top;
4      void init() { memset(fa, -1, sizeof fa);
5          memset(sz, 0, sizeof sz); top = 0; }

```



```

5 int findset(int x) { while (fa[x] != -1)
6   x = fa[x]; return x; }
7 bool join(int x, int y) {
8   x = findset(x); y = findset(y);
9   if (x == y) return false;
10  if (sz[x] > sz[y]) swap(x, y);
11  undo[top++] = x;
12  fa[x] = y;
13  sz[y] += sz[x] + 1;
14  return true;
15 }
16 inline int checkpoint() { return top; }
17 void rewind(int t) {
18   while (top > t) {
19     int x = undo[--top];
20     sz[fa[x]] -= sz[x] + 1;
21     fa[x] = -1;
22   }
23 }

```

3 Math

3.1 Multiplication, Powers

```

1 LL mul(LL u, LL v, LL p) {
2   return (u * v - LL((long double) u * v /
3     p) * p + p) % p;
4 }
5 LL mul(LL u, LL v, LL p) { // better constant
6   LL t = u * v - LL((long double) u * v / p
7     ) * p;
8   return t < 0 ? t + p : t;
9 }
10 LL bin(LL x, LL n, LL MOD) {
11   n %= (MOD - 1); // if MOD is prime
12   LL ret = MOD - 1;
13   for (x %= MOD; n; n >>= 1, x = mul(x, x,
14     MOD))
15     if (n & 1) ret = mul(ret, x, MOD);
16 }

```

3.2 Matrix Power

```

1 struct Mat {
2   static const LL M = 2;
3   LL v[M][M];
4   Mat() { memset(v, 0, sizeof v); }
5   void eye() { FOR (i, 0, M) v[i][i] = 1; }
6   LL* operator [] (LL x) { return v[x]; }
7   const LL* operator [] (LL x) const {
8     return v[x]; }
9   Mat operator * (const Mat& B) {
10    const Mat& A = *this;
11    Mat ret;
12    FOR (k, 0, M)
13      FOR (i, 0, M) if (A[i][k])
14        FOR (j, 0, M)

```

```

14      ret[i][j] = (ret[i][j] +
15        A[i][k] * B[k][j]) %
16        MOD;
17    }
18    return ret;
19  }
20  Mat pow(LL n) const {
21    Mat A = *this, ret; ret.eye();
22    for (; n >>= 1, A = A * A;
23      if (n & 1) ret = ret * A;
24      return ret;
25  }
26  Mat operator + (const Mat& B) {
27    const Mat& A = *this;
28    Mat ret;
29    FOR (i, 0, M)
30      FOR (j, 0, M)
31        ret[i][j] = (A[i][j] + B[i][j]) % MOD;
32    return ret;
33  }
34  void prt() const {
35    FOR (i, 0, M)
36      FOR (j, 0, M)
37        printf("%lld%c", (*this)[i][j], j == M - 1 ? '\n' : ' ');
38  }
39 }

```

3.3 Sieve

```

1 const LL p_max = 1E5 + 100;
2 LL phi[p_max];
3 void get_phi() {
4   phi[1] = 1;
5   static bool vis[p_max];
6   static LL prime[p_max], p_sz, d;
7   FOR (i, 2, p_max) {
8     if (!vis[i]) {
9       prime[p_sz++] = i;
10      phi[i] = i - 1;
11    }
12    for (LL j = 0; j < p_sz && (d = i *
13      prime[j]) < p_max; ++j) {
14      vis[d] = 1;
15      if (i % prime[j] == 0) {
16        phi[d] = phi[i] * prime[j];
17        break;
18      }
19      else phi[d] = phi[i] * (prime[j] - 1);
20    }
21  }
22 }
23 // mobius
24 const LL p_max = 1E5 + 100;
25 LL mu[p_max];
26 void get_mu() {
27   mu[1] = 1;
28   static bool vis[p_max];
29   static LL prime[p_max], p_sz, d;
30   mu[1] = 1;
31   FOR (i, 2, p_max) {
32     if (!vis[i]) {

```

```

32     prime[p_sz++] = i;
33     mu[i] = -1;
34   }
35   for (LL j = 0; j < p_sz && (d = i *
36     prime[j]) < p_max; ++j) {
37     vis[d] = 1;
38     if (i % prime[j] == 0) {
39       mu[d] = 0;
40       break;
41     }
42     else mu[d] = -mu[i];
43   }
44 }
45 // min_25
46 namespace min25 {
47   const int M = 1E6 + 100;
48   LL B, N;
49   // g(x)
50   inline LL pg(LL x) { return 1; }
51   inline LL ph(LL x) { return x % MOD; }
52   // Sum[g(i), {x, 2, x}]
53   inline LL psg(LL x) { return x % MOD - 1; }
54   inline LL psh(LL x) {
55     static LL inv2 = (MOD + 1) / 2;
56     x = x % MOD;
57     return x * (x + 1) % MOD * inv2 % MOD - 1;
58   }
59   // f(pp=p^k)
60   inline LL fpk(LL p, LL e, LL pp) { return
61     (pp - pp / p) % MOD; }
62   // f(p) = fgh(g(p), h(p))
63   inline LL fgh(LL g, LL h) { return h - g; }
64 }
65 LL pr[M], pc, sg[M], sh[M];
66 void get_prime(LL n) {
67   static bool vis[M]; pc = 0;
68   FOR (i, 2, n + 1) {
69     if (!vis[i]) {
70       pr[pc++] = i;
71       sg[pc] = (sg[pc - 1] + pg(i)) % MOD;
72       sh[pc] = (sh[pc - 1] + ph(i)) % MOD;
73     }
74     FOR (j, 0, pc) {
75       if (pr[j] * i > n) break;
76       vis[pr[j] * i] = 1;
77       if (i % pr[j] == 0) break;
78     }
79   }
80   LL w[M];
81   LL id1[M], id2[M], h[M], g[M];
82   inline LL id(LL x) { return x <= B ? id1[x] : id2[N / x]; }
83   LL go(LL x, LL k) {
84     if (x <= 1 || (k >= 0 && pr[k] > x))
85       return 0;
86     LL t = id(x);
87     LL ans = fgh((g[t] - sg[k + 1]), (h[t] - sh[k + 1]));
88     FOR (i, k + 1, pc) {

```

```

88     LL p = pr[i];
89     if (p * p > x) break;
90     ans -= fgh(pg(p), ph(p));
91     for (LL pp = p, e = 1; pp <= x;
92         ++e, pp = pp * p)
93         ans += fpk(p, e, pp) * (1 +
94             go(x / pp, i)) % MOD;
95     }
96     return ans % MOD;
97 }
98 LL solve(LL _N) {
99     N = _N;
100     B = sqrt(N + 0.5);
101     get_prime(B);
102     int sz = 0;
103     for (LL l = 1, v, r; l <= N; l = r +
104         1) {
105         v = N / l; r = N / v;
106         w[sz] = v; g[sz] = psg(v); h[sz]
107         = psh(v);
108         if (v <= B) id1[v] = sz; else id2
109         [r] = sz;
110         sz++;
111     }
112     FOR (k, 0, pc) {
113         LL p = pr[k];
114         FOR (i, 0, sz) {
115             LL v = w[i]; if (p * p > v)
116             break;
117             LL t = id(v / p);
118             g[i] = (g[i] - (g[t] - sg[k])
119                 * pg(p)) % MOD;
120             h[i] = (h[i] - (h[t] - sh[k])
121                 * ph(p)) % MOD;
122         }
123     }
124     return (go(N, -1) % MOD + MOD + 1) %
125     MOD;
126 }
127 // see cheatsheet for instructions
128 namespace dujiao {
129     const int M = 5E6;
130     LL f[M] = {0, 1};
131     void init() {
132         static bool vis[M];
133         static LL pr[M], p_sz, d;
134         FOR (i, 2, M) {
135             if (!vis[i]) { pr[p_sz++] = i; f[
136                 i] = -1; }
137             FOR (j, 0, p_sz) {
138                 if ((d = pr[j] * i) >= M)
139                 break;
140                 vis[d] = 1;
141                 if (i % pr[j] == 0) {
142                     f[d] = 0;
143                     break;
144                 } else f[d] = -f[i];
145             }
146         }
147         FOR (i, 2, M) f[i] += f[i - 1];
148     }
149     inline LL s_fg(LL n) { return 1; }
150     inline LL s_g(LL n) { return n; }
151     LL N, rd[M];

```

```

143     bool vis[M];
144     LL go(LL n) {
145         if (n < M) return f[n];
146         LL id = N / n;
147         if (vis[id]) return rd[id];
148         vis[id] = true;
149         LL& ret = rd[id] = s_fg(n);
150         for (LL l = 2, v, r; l <= n; l = r +
151             1) {
152             v = n / l; r = n / v;
153             ret -= (s_g(r) - s_g(l - 1)) * go
154             (v);
155         }
156         return ret;
157     }
158     LL solve(LL n) {
159         N = n;
160         memset(vis, 0, sizeof vis);
161         return go(n);
162     }

```

3.4 Prime Test

```

1     bool checkQ(LL a, LL n) {
2         if (n == 2 || a >= n) return 1;
3         if (n == 1 || !(n & 1)) return 0;
4         LL d = n - 1;
5         while (!(d & 1)) d >>= 1;
6         LL t = bin(a, d, n); // usually needs
7         mul-on-LL
8         while (d != n - 1 && t != 1 && t != n -
9             1) {
10             t = mul(t, t, n);
11             d <<= 1;
12         }
13         return t == n - 1 || d & 1;
14     }
15     bool primeQ(LL n) {
16         static vector<LL> t = {2, 325, 9375,
17             28178, 450775, 9780504, 1795265022};
18         if (n <= 1) return false;
19         for (LL k: t) if (!checkQ(k, n)) return
20         false;
21         return true;
22     }

```

3.5 Pollard-Rho

```

1     mt19937 mt(time(0));
2     LL pollard_rho(LL n, LL c) {
3         LL x = uniform_int_distribution<LL>(1, n
4             - 1)(mt), y = x;
5         auto f = [&](LL v) { LL t = mul(v, v, n)
6             + c; return t < n ? t : t - n; };
7         while (1) {
8             x = f(x); y = f(f(y));
9             if (x == y) return n;
10            LL d = gcd(abs(x - y), n);
11            if (d != 1) return d;
12        }

```

```

11     }
12     LL fac[100], fcnt;
13     void get_fac(LL n, LL cc = 19260817) {
14         if (n == 4) { fac[fcnt++] = 2; fac[fcnt
15             ++] = 2; return; }
16         if (primeQ(n)) { fac[fcnt++] = n; return;
17             }
18         LL p = n;
19         while (p == n) p = pollard_rho(n, --cc);
20         get_fac(p); get_fac(n / p);
21     }

```

3.6 Berlekamp-Massey

```

1     namespace BerlekampMassey {
2         inline void up(LL& a, LL b) { (a += b) %=
3             MOD; }
4         V mul(const V& a, const V& b, const V& m,
5             int k) {
6             V r; r.resize(2 * k - 1);
7             FOR (i, 0, k) FOR (j, 0, k) up(r[i +
8                 j], a[i] * b[j]);
9             FORD (i, k - 2, -1) {
10                 FOR (j, 0, k) up(r[i + j], r[i +
11                     k] * m[j]);
12                 r.pop_back();
13             }
14             return r;
15         }
16         V pow(LL n, const V& m) {
17             int k = (int) m.size() - 1; assert (m
18                 [k] == -1 || m[k] == MOD - 1);
19             V r(k, x(k); r[0] = x[1] = 1;
20             for (; n; n >>= 1, x = mul(x, x, m, k
21                 ))
22                 if (n & 1) r = mul(x, r, m, k);
23             return r;
24         }
25         LL go(const V& a, const V& x, LL n) {
26             // a: (-1, a1, a2, ..., ak).reverse
27             // x: x1, x2, ..., xk
28             // x[n] = sum[a[i]*x[n-i], {i, 1, k}]
29             int k = (int) a.size() - 1;
30             if (n <= k) return x[n - 1];
31             if (a.size() == 2) return x[0] * bin(
32                 a[0], n - 1, MOD) % MOD;
33             V r = pow(n - 1, a);
34             LL ans = 0;
35             FOR (i, 0, k) up(ans, r[i] * x[i]);
36             return (ans + MOD) % MOD;
37         }
38         V BM(const V& x) {
39             V a = {-1}, b = {233}, t;
40             FOR (i, 1, x.size()) {
41                 b.push_back(0);
42                 LL d = 0, la = a.size(), lb = b.
43                     size();
44                 FOR (j, 0, la) up(d, a[j] * x[i -
45                     la + 1 + j]);
46                 if (d == 0) continue;
47                 t.clear(); for (auto& v: b) t.
48                     push_back(d * v % MOD);
49                 FOR (i, 0, la - lb) t.push_back
50                     (0);

```

```

40     lb = max(la, lb);
41     FOR (j, 0, la) up(t[lb - 1 - j],
42         a[lb - 1 - j]);
43     if (lb > la) {
44         b.swap(a);
45         LL inv = -get_inv(d, MOD);
46         for (auto& v: b) v = v * inv
47             % MOD;
48     }
49     a.swap(t);
50     for (auto& v: a) up(v, MOD);
51     return a;
52 }

```

3.7 Extended Euclidean

```

1 LL ex_gcd(LL a, LL b, LL &x, LL &y) {
2     if (b == 0) { x = 1; y = 0; return a; }
3     LL ret = ex_gcd(b, a % b, y, x);
4     y -= a / b * x;
5     return ret;
6 }
7 ///////////////////////////////////////////////////
8 inline int ctz(LL x) { return __builtin_ctzll
9     (x); }
10 LL gcd(LL a, LL b) {
11     if (!a) return b; if (!b) return a;
12     int t = ctz(a | b);
13     a >>= ctz(a);
14     do {
15         b >>= ctz(b);
16         if (a > b) swap(a, b);
17         b -= a;
18     } while (b);
19     return a << t;

```

3.8 Inverse

```

1 // if p is prime
2 inline LL get_inv(LL x, LL p) { return bin(x,
3     p - 2, p); }
4 // if p is not prime
5 LL get_inv(LL a, LL M) {
6     static LL x, y;
7     assert(exgcd(a, M, x, y) == 1);
8     return (x % M + M) % M;
9 }
10 ///////////////////////////////////////////////////
11 LL inv[N];
12 void inv_init(LL n, LL p) {
13     inv[1] = 1;
14     FOR (i, 2, n)
15         inv[i] = (p - p / i) * inv[p % i] % p;
16 }
17 ///////////////////////////////////////////////////
18 LL invf[M], fac[M] = {1};
19 void fac_inv_init(LL n, LL p) {

```

```

19     FOR (i, 1, n)
20         fac[i] = i * fac[i - 1] % p;
21     invf[n - 1] = bin(fac[n - 1], p - 2, p);
22     FORD (i, n - 2, -1)
23         invf[i] = invf[i + 1] * (i + 1) % p;
24 }

```

3.9 Binomial Numbers

```

1 inline LL C(LL n, LL m) { // n >= m >= 0
2     return n < m || m < 0 ? 0 : fac[n] * invf
3         [m] % MOD * invf[n - m] % MOD;
4 }
5 // The following code reverses n and m
6 LL C(LL n, LL m) { // m >= n >= 0
7     if (m - n < n) n = m - n;
8     if (n < 0) return 0;
9     LL ret = 1;
10    FOR (i, 1, n + 1)
11        ret = ret * (m - n + i) % MOD * bin(i
12            , MOD - 2, MOD) % MOD;
13    return ret;
14 }
15 LL Lucas(LL n, LL m) { // m >= n >= 0
16    return m ? C(n % MOD, m % MOD) * Lucas(n
17        / MOD, m / MOD) % MOD : 1;
18 }
19 // precalculations
20 LL C[M][M];
21 void init_C(int n) {
22     FOR (i, 0, n) {
23         C[i][0] = C[i][i] = 1;
24         FOR (j, 1, i)
25             C[i][j] = (C[i - 1][j] + C[i -
26                 1][j - 1]) % MOD;
27     }
28 }

```

3.10 NTT, FFT, FWT

```

1 // NTT
2 LL wn[N << 2], rev[N << 2];
3 int NTT_init(int n) {
4     int step = 0; int n = 1;
5     for (; n < n_; n <= 1) ++step;
6     FOR (i, 1, n)
7         rev[i] = (rev[i >> 1] >> 1) | ((i &
8             1) << (step - 1));
9     int g = bin(G, (MOD - 1) / n, MOD);
10    wn[0] = 1;
11    for (int i = 1; i <= n; ++i)
12        wn[i] = wn[i - 1] * g % MOD;
13    return n;
14 }
15 void NTT(LL a[], int n, int f) {
16     FOR (i, 0, n) if (i < rev[i])
17         std::swap(a[i], a[rev[i]]);
18     for (int k = 1; k < n; k <= 1) {
19         for (int i = 0; i < n; i += (k << 1))
20             int t = n / (k << 1);

```

```

20     FOR (j, 0, k) {
21         LL w = f == 1 ? wn[t * j] :
22             wn[n - t * j];
23         LL x = a[i + j];
24         LL y = a[i + j + k] * w % MOD;
25         a[i + j] = (x + y) % MOD;
26         a[i + j + k] = (x - y + MOD)
27             % MOD;
28     }
29     if (f == -1) {
30         LL ninv = get_inv(n, MOD);
31         FOR (i, 0, n)
32             a[i] = a[i] * ninv % MOD;
33     }
34 }
35 // FFT
36 // n needs to be power of 2
37 typedef double LD;
38 const LD PI = acos(-1);
39 struct C {
40     LD r, i;
41     C(LD r = 0, LD i = 0): r(r), i(i) {}
42 };
43 C operator + (const C& a, const C& b) {
44     return C(a.r + b.r, a.i + b.i);
45 }
46 C operator - (const C& a, const C& b) {
47     return C(a.r - b.r, a.i - b.i);
48 }
49 C operator * (const C& a, const C& b) {
50     return C(a.r * b.r - a.i * b.i, a.r * b.i
51         + a.i * b.r);
52 }
53 void FFT(C x[], int n, int p) {
54     for (int i = 0, t = 0; i < n; ++i) {
55         if (i > t) swap(x[i], x[t]);
56         for (int j = n >> 1; (t ^= j) < j; j
57             >>= 1);
58     }
59     for (int h = 2; h <= n; h <= 1) {
60         C wn(cos(p * 2 * PI / h), sin(p * 2 *
61             PI / h));
62         for (int i = 0; i < n; i += h) {
63             C w(1, 0), u;
64             for (int j = i, k = h >> 1; j < i
65                 + k; ++j) {
66                 u = x[j + k] * w;
67                 x[j + k] = x[j] - u;
68                 x[j] = x[j] + u;
69                 w = w * wn;
70             }
71         }
72     }
73     if (p == -1)
74         FOR (i, 0, n)
75             x[i].r /= n;
76 }
77 void conv(C a[], C b[], int n) {
78     FFT(a, n, 1);
79     FFT(b, n, 1);
80     FOR (i, 0, n)
81         a[i] = a[i] * b[i];
82     FFT(a, n, -1);

```

```

79 }
80
81 // FWT
82 // C_k = \sum_{i \oplus j = k} A_i B_j
83 template<typename T>
84 void fwt(LL a[], int n, T f) {
85     for (int d = 1; d < n; d *= 2)
86         for (int i = 0, t = d * 2; i < n; i
87              += t)
88             FOR (j, 0, d)
89                 f(a[i + j], a[i + j + d]);
90 }
91
92 void AND(LL& a, LL& b) { a += b; }
93 void OR(LL& a, LL& b) { b += a; }
94 void XOR (LL& a, LL& b) {
95     LL x = a, y = b;
96     a = (x + y) % MOD;
97     b = (x - y + MOD) % MOD;
98 }
99 void rAND(LL& a, LL& b) { a -= b; }
100 void rOR(LL& a, LL& b) { b -= a; }
101 void rXOR(LL& a, LL& b) {
102     static LL INV2 = (MOD + 1) / 2;
103     LL x = a, y = b;
104     a = (x + y) * INV2 % MOD;
105     b = (x - y + MOD) * INV2 % MOD;
106 }
107 /*
108 FWT subset convolution
109 a[popcount(x)][x] = A[x]
110 b[popcount(x)][x] = B[x]
111 fwt(a[i]) fwt(b[i])
112 c[i + j][x] += a[i][x] * b[j][x]
113 rfwf(c[i])
114 ans[x] = c[popcount(x)][x]
115 */

```

3.11 Simpson's Numerical Integration

```

1 LD simpson(LD l, LD r) {
2     LD c = (l + r) / 2;
3     return (f(l) + 4 * f(c) + f(r)) * (r - l)
4         / 6;
5 }
6
7 LD asr(LD l, LD r, LD eps, LD S) {
8     LD m = (l + r) / 2;
9     LD L = simpson(l, m), R = simpson(m, r);
10    if (fabs(L + R - S) < 15 * eps) return L
11        + R + (L + R - S) / 15;
12    return asr(l, m, eps / 2, L) + asr(m, r,
13        eps / 2, R);
14 }
15
16 LD asr(LD l, LD r, LD eps) { return asr(l, r,
17     eps, simpson(l, r)); }

```

3.12 Gauss Elimination

```

1 // n equations, m variables
2 // a is an n x (m + 1) augmented matrix
3 // free is an indicator of free variable
4 // return the number of free variables, -1
5 // for "404"
6
7 int n, m;
8 LD a[maxn][maxn], x[maxn];
9 bool free_x[maxn];
10 inline int sgn(LD x) { return (x > eps) - (x
11     < -eps); }
12 int gauss(LD a[maxn][maxn], int n, int m) {
13     memset(free_x, 1, sizeof free_x); memset(x,
14         0, sizeof x);
15     int r = 0, c = 0;
16     while (r < n && c < m) {
17         int m_r = r;
18         FOR (i, r + 1, n)
19             if (fabs(a[i][c]) > fabs(a[m_r][c]))
20                 m_r = i;
21         if (m_r != r)
22             swap(a[r][c], a[m_r][c]);
23         if (!sgn(a[r][c])) {
24             a[r][c] = 0; ++c;
25             continue;
26         }
27         FOR (i, r + 1, n)
28             if (a[i][c]) {
29                 LD t = a[i][c] / a[r][c];
30                 FOR (j, c, m + 1) a[i][j] -= a[r][j]
31                     * t;
32             }
33         ++r; ++c;
34     }
35     FOR (i, r, n)
36         if (sgn(a[i][m])) return -1;
37     if (r < m) {
38         FORD (i, r - 1, -1) {
39             int f_cnt = 0, k = -1;
40             FOR (j, 0, m)
41                 if (sgn(a[i][j]) && free_x[j]) {
42                     ++f_cnt; k = j;
43                 }
44             if (f_cnt > 0) continue;
45             LD s = a[i][m];
46             FOR (j, 0, m)
47                 if (j != k) s -= a[i][j] * x[j];
48             x[k] = s / a[i][k];
49             free_x[k] = 0;
50         }
51         return m - r;
52     }
53     FORD (i, m - 1, -1) {
54         LD s = a[i][m];
55         FOR (j, i + 1, m)
56             s -= a[i][j] * x[j];
57         x[i] = s / a[i][i];
58     }
59     return 0;
60 }

```

3.13 Factor Decomposition

```

1 LL factor[30], f_sz, factor_exp[30];
2 void get_factor(LL x) {
3     f_sz = 0;
4     LL t = sqrt(x + 0.5);
5     for (LL i = 0; pr[i] <= t; ++i)
6         if (x % pr[i] == 0) {
7             factor_exp[f_sz] = 0;
8             while (x % pr[i] == 0) {
9                 x /= pr[i];
10                ++factor_exp[f_sz];
11            }
12            factor[f_sz++] = pr[i];
13        }
14    if (x > 1) {
15        factor_exp[f_sz] = 1;
16        factor[f_sz++] = x;
17    }
18 }

```

3.14 Primitive Root

```

1 LL find_smallest_primitive_root(LL p) {
2     // p should be a prime
3     get_factor(p - 1);
4     FOR (i, 2, p) {
5         bool flag = true;
6         FOR (j, 0, f_sz)
7             if (bin(i, (p - 1) / factor[j], p
8                 ) == 1) {
9                 flag = false;
10                break;
11            }
12        if (flag) return i;
13    }
14    assert(0); return -1;
15 }

```

3.15 Quadratic Residue

```

1 LL q1, q2, w;
2 struct P { // x + y * sqrt(w)
3     LL x, y;
4 };
5 P pmul(const P& a, const P& b, LL p) {
6     P res;
7     res.x = (a.x * b.x + a.y * b.y % p * w) %
8         p;
9     res.y = (a.x * b.y + a.y * b.x) % p;
10    return res;
11 }
12 P bin(P x, LL n, LL MOD) {
13     P ret = {1, 0};
14     for (; n; n >>= 1, x = pmul(x, x, MOD))
15         if (n & 1) ret = pmul(ret, x, MOD);
16     return ret;
17 }
18 LL Legendre(LL a, LL p) { return bin(a, (p -
19     1) >> 1, p); }
20 LL equation_solve(LL b, LL p) {
21     if (p == 2) return 1;
22     if ((Legendre(b, p) + 1) % p == 0)

```

```

21     return -1;
22 LL a;
23 while (true) {
24     a = rand() % p;
25     w = ((a * a - b) % p + p) % p;
26     if ((Legendre(w, p) + 1) % p == 0)
27         break;
28 }
29 return bin({a, 1}, (p + 1) >> 1, p).x;
30 }
31 // Given a and prime p, find x such that x*x=
32 // a(mod p)
33 int main() {
34     LL a, p; cin >> a >> p;
35     a = a % p;
36     LL x = equation_solve(a, p);
37     if (x == -1) {
38         puts("No root");
39     } else {
40         LL y = p - x;
41         if (x == y) cout << x << endl;
42         else cout << min(x, y) << " " << max(
43             x, y) << endl;
44     }
45 }

```

3.16 Chinese Remainder Theorem

```

1 LL CRT(LL *m, LL *r, LL n) {
2     if (!n) return 0;
3     LL M = m[0], R = r[0], x, y, d;
4     FOR (i, 1, n) {
5         d = ex_gcd(M, m[i], x, y);
6         if ((r[i] - R) % d) return -1;
7         x = (r[i] - R) / d * x % (m[i] / d);
8         R += x * M;
9         M = M / d * m[i];
10        R %= M;
11    }
12    return R >= 0 ? R : R + M;
13 }

```

3.17 Bernoulli Numbers

```

1 namespace Bernoulli {
2     LL inv[M] = {-1, 1};
3     LL C[M][M];
4     void init();
5     LL B[M] = {1};
6     void init() {
7         inv_init(M, MOD);
8         init_C(M);
9         FOR (i, 1, M - 1) {
10             LL& s = B[i] = 0;
11             FOR (j, 0, i)
12                 s += C[i + 1][j] * B[j] % MOD;
13             s = (s % MOD * -inv[i + 1] % MOD
14                 + MOD) % MOD;
15         }
16     }
17 }

```

```

16 LL p[M] = {1};
17 LL go(LL n, LL k) {
18     n %= MOD;
19     if (k == 0) return n;
20     FOR (i, 1, k + 2)
21         p[i] = p[i - 1] * (n + 1) % MOD;
22     LL ret = 0;
23     FOR (i, 1, k + 2)
24         ret += C[k + 1][i] * B[k + 1 - i]
25             % MOD * p[i] % MOD;
26     ret = ret % MOD * inv[k + 1] % MOD;
27     return ret;
28 }

```

3.18 Simplex Method

```

1 // x = 0 should satisfy the constraints
2 // initialize v to be 0
3 // n is dimension of vector, m is number of
4 // constraints
5 // min{ b x } / max { c x }
6 // A x >= c / A x <= b
7 // x >= 0
8 namespace lp {
9     int n, m;
10    double a[M][N], b[M], c[N], v;
11
12    void pivot(int l, int e) {
13        b[l] /= a[l][e];
14        FOR (j, 0, n) if (j != e) a[l][j] /=
15            a[l][e];
16        a[l][e] = 1 / a[l][e];
17
18        FOR (i, 0, m)
19            if (i != l && fabs(a[i][e]) > 0)
20                {
21                    b[i] -= a[i][e] * b[l];
22                    FOR (j, 0, n)
23                        if (j != e) a[i][j] -= a[
24                            i][e] * a[l][j];
25                    a[i][e] = -a[i][e] * a[l][e];
26                }
27        v += c[e] * b[l];
28        FOR (j, 0, n) if (j != e) c[j] -= c[e]
29            * a[l][j];
30        c[e] = -c[e] * a[l][e];
31    }
32
33    double simplex() {
34        while (1) {
35            v = 0;
36            int e = -1, l = -1;
37            FOR (i, 0, n) if (c[i] > eps) { e
38                = i; break; }
39            if (e == -1) return v;
40            double t = INF;
41            FOR (i, 0, m)
42                if (a[i][e] > eps && t > b[i]
43                    / a[i][e]) {
44                    t = b[i] / a[i][e];
45                    l = i;
46                }
47            if (l == -1) return INF;
48            pivot(l, e);
49        }
50    }
51 }

```

```

41     }
42 }
43 }

```

3.19 BSGS

```

1 // p is a prime
2 LL BSGS(LL a, LL b, LL p) { // a^x = b (mod p)
3     a %= p;
4     if (!a && !b) return 1;
5     if (!a) return -1;
6     static map<LL, LL> mp; mp.clear();
7     LL m = sqrt(p + 1.5);
8     LL v = 1;
9     FOR (i, 1, m + 1) {
10        v = v * a % p;
11        mp[v * b % p] = i;
12    }
13    LL vv = v;
14    FOR (i, 1, m + 1) {
15        auto it = mp.find(vv);
16        if (it != mp.end()) return i * m - it
17            ->second;
18        vv = vv * v % p;
19    }
20    return -1;
21 }
22 // p can be not a prime
23 LL exBSGS(LL a, LL b, LL p) { // a^x = b (mod
24     p)
25     a %= p; b %= p;
26     if (a == 0) return b > 1 ? -1 : b == 0 &&
27         p != 1;
28     LL c = 0, q = 1;
29     while (1) {
30         LL g = __gcd(a, p);
31         if (g == 1) break;
32         if (b == 1) return c;
33         if (b % g) return -1;
34         ++c; b /= g; p /= g; q = a / g * q %
35             p;
36     }
37     static map<LL, LL> mp; mp.clear();
38     LL m = sqrt(p + 1.5);
39     LL v = 1;
40     FOR (i, 1, m + 1) {
41        v = v * a % p;
42        mp[v * b % p] = i;
43    }
44    FOR (i, 1, m + 1) {
45        q = q * v % p;
46        auto it = mp.find(q);
47        if (it != mp.end()) return i * m - it
48            ->second + c;
49    }
50    return -1;
51 }

```

4 Graph Theory

4.1 LCA

```

1 void dfs(int u, int fa) {
2     pa[u][0] = fa; dep[u] = dep[fa] + 1;
3     FOR (i, 1, SP) pa[u][i] = pa[pa[u][i - 1]][i - 1];
4     for (int& v: G[u]) {
5         if (v == fa) continue;
6         dfs(v, u);
7     }
8 }
9 int lca(int u, int v) {
10    if (dep[u] < dep[v]) swap(u, v);
11    int t = dep[u] - dep[v];
12    FOR (i, 0, SP) if (t & (1 << i)) u = pa[u][i];
13    FOR (i, SP - 1, -1) {
14        int uu = pa[u][i], vv = pa[v][i];
15        if (uu != vv) { u = uu; v = vv; }
16    }
17    return u == v ? u : pa[u][0];
18 }

```

4.2 Maximum Flow

```

1 struct E {
2     int to, cp;
3     E(int to, int cp): to(to), cp(cp) {}
4 };
5
6 struct Dinic {
7     static const int M = 1E5 * 5;
8     int m, s, t;
9     vector<E> edges;
10    vector<int> G[M];
11    int d[M];
12    int cur[M];
13    void init(int n, int s, int t) {
14        this->s = s; this->t = t;
15        for (int i = 0; i <= n; i++) G[i].clear();
16        edges.clear(); m = 0;
17    }
18    void addedge(int u, int v, int cap) {
19        edges.emplace_back(v, cap);
20        edges.emplace_back(u, 0);
21        G[u].push_back(m++);
22        G[v].push_back(m++);
23    }
24    bool BFS() {
25        memset(d, 0, sizeof d);
26        queue<int> Q;
27        Q.push(s); d[s] = 1;
28        while (!Q.empty()) {
29            int x = Q.front(); Q.pop();
30            for (int& i: G[x]) {
31                E& e = edges[i];
32                if (!d[e.to] && e.cp > 0) {
33                    d[e.to] = d[x] + 1;
34                    Q.push(e.to);
35                }
36            }
37        }
38    }
39    int DFS(int u, int cp) {
40        if (u == t || !cp) return cp;
41        int tmp = cp, f;
42        for (int& i = cur[u]; i < G[u].size(); i++) {
43            E& e = edges[G[u][i]];
44            if (d[u] + 1 == d[e.to]) {
45                f = DFS(e.to, min(cp, e.cp));
46                e.cp -= f;
47                edges[G[u][i] ^ 1].cp += f;
48                cp -= f;
49                if (!cp) break;
50            }
51        }
52        return tmp - cp;
53    }
54    int go() {
55        int flow = 0;
56        while (BFS()) {
57            memset(cur, 0, sizeof cur);
58            flow += DFS(s, INF);
59        }
60        return flow;
61    }
62 } DC;

```

4.3 Minimum Cost Maximum Flow

```

1 struct E {
2     int from, to, cp, v;
3     E() {}
4     E(int f, int t, int cp, int v) : from(f), to(t), cp(cp), v(v) {}
5 };
6 struct MCMF {
7     int n, m, s, t;
8     vector<E> edges;
9     vector<int> G[maxn];
10    bool inq[maxn];
11    int d[maxn]; // shortest path
12    int p[maxn]; // the last edge id of the path from s to i
13    int a[maxn]; // least remaining capacity from s to i
14    void init(int _n, int _s, int _t) {}
15    void addedge(int from, int to, int cap, int cost) {
16        edges.emplace_back(from, to, cap, cost);
17        edges.emplace_back(to, from, 0, -cost);
18        G[from].push_back(m++);
19        G[to].push_back(m++);
20    }
21    bool BellmanFord(int &flow, int &cost) {
22        FOR (i, 0, n + 1) d[i] = INF;
23        memset(inq, 0, sizeof inq);
24        d[s] = 0, a[s] = INF, inq[s] = true;
25        queue<int> Q; Q.push(s);
26        while (!Q.empty()) {

```

```

27            int u = Q.front(); Q.pop();
28            inq[u] = false;
29            for (int& idx: G[u]) {
30                E& e = edges[idx];
31                if (e.cp && d[e.to] > d[u] + e.v) {
32                    d[e.to] = d[u] + e.v;
33                    p[e.to] = idx;
34                    a[e.to] = min(a[u], e.cp);
35                    if (!inq[e.to]) {
36                        Q.push(e.to);
37                        inq[e.to] = true;
38                    }
39                }
40            }
41            if (d[t] == INF) return false;
42            flow += a[t];
43            cost += a[t] * d[t];
44            int u = t;
45            while (u != s) {
46                edges[p[u]].cp -= a[t];
47                edges[p[u] ^ 1].cp += a[t];
48                u = edges[p[u]].from;
49            }
50            return true;
51        }
52    }
53    int go() {
54        int flow = 0, cost = 0;
55        while (BellmanFord(flow, cost))
56            return cost;
57    }
58 } MM;

```

4.4 Path Intersection on Trees

```

1 int intersection(int x, int y, int xx, int yy) {
2     int t[4] = {lca(x, xx), lca(x, yy), lca(y, xx), lca(y, yy)};
3     sort(t, t + 4);
4     int r = lca(x, y), rr = lca(xx, yy);
5     if (dep[t[0]] < min(dep[r], dep[rr]) || dep[t[2]] < max(dep[r], dep[rr]))
6         return 0;
7     int tt = lca(t[2], t[3]);
8     int ret = 1 + dep[t[2]] + dep[t[3]] - dep[tt] * 2;
9     return ret;
10 }

```

4.5 Centroid Decomposition (Divide-Conquer)

```

1 int get_rt(int u) {
2     static int q[N], fa[N], sz[N], mx[N];
3     int p = 0, cur = -1;
4     q[p++] = u; fa[u] = -1;

```



```

5   while (++cur < p) {
6       u = q[cur]; mx[u] = 0; sz[u] = 1;
7       for (int& v: G[u])
8           if (!vis[v] && v != fa[u]) fa[q[p
9           ++] = v] = u;
10  }
11  FORD (i, p - 1, -1) {
12      u = q[i];
13      mx[u] = max(mx[u], p - sz[u]);
14      if (mx[u] * 2 <= p) return u;
15      sz[fa[u]] += sz[u];
16      mx[fa[u]] = max(mx[fa[u]], sz[u]);
17  }
18  assert(0);
19  }
20  void dfs(int u) {
21      u = get_rt(u);
22      vis[u] = true;
23      get_dep(u, -1, 0);
24      // ...
25      for (E& e: G[u]) {
26          int v = e.to;
27          if (vis[v]) continue;
28          // ...
29          dfs(v);
30      }
31  }
32  ///////////////////////////////////////////////////
33  // dynamic divide and conquer
34  ///////////////////////////////////////////////////
35  const int maxn = 15E4 + 100, INF = 1E9;
36  struct E {
37      int to, d;
38  };
39  vector<E> G[maxn];
40  int n, Q, w[maxn];
41  LL A, ans;
42  bool vis[maxn];
43  int sz[maxn];
44  int get_rt(int u) {
45      static int q[N], fa[N], sz[N], mx[N];
46      int p = 0, cur = -1;
47      q[p++] = u; fa[u] = -1;
48      while (++cur < p) {
49          u = q[cur]; mx[u] = 0; sz[u] = 1;
50          for (int& v: G[u])
51              if (!vis[v] && v != fa[u]) fa[q[p
52              ++] = v] = u;
53      }
54      FORD (i, p - 1, -1) {
55          u = q[i];
56          mx[u] = max(mx[u], p - sz[u]);
57          if (mx[u] * 2 <= p) return u;
58          sz[fa[u]] += sz[u];
59          mx[fa[u]] = max(mx[fa[u]], sz[u]);
60      }
61      assert(0);
62  }
63  }
64  int dep[maxn], md[maxn];
65  void get_dep(int u, int fa, int d) {
66      dep[u] = d; md[u] = 0;

```

```

69  for (E& e: G[u]) {
70      int v = e.to;
71      if (vis[v] || v == fa) continue;
72      get_dep(v, u, d + e.d);
73      md[u] = max(md[u], md[v] + 1);
74  }
75  }
76  struct P {
77      int w;
78      LL s;
79  };
80  using VP = vector<P>;
81  struct R {
82      VP *rt, *rt2;
83      int dep;
84  };
85  VP pool[maxn << 1], *pit = pool;
86  vector<R> tr[maxn];
87  void go(int u, int fa, VP* rt, VP* rt2) {
88      tr[u].push_back({rt, rt2, dep[u]});
89      for (E& e: G[u]) {
90          int v = e.to;
91          if (v == fa || vis[v]) continue;
92          go(v, u, rt, rt2);
93      }
94  }
95  }
96  void dfs(int u) {
97      u = get_rt(u);
98      vis[u] = true;
99      get_dep(u, -1, 0);
100     VP* rt = pit++; tr[u].push_back({rt,
101     nullptr, 0});
102     for (E& e: G[u]) {
103         int v = e.to;
104         if (vis[v]) continue;
105         go(v, u, rt, pit++);
106         dfs(v);
107     }
108 }
109 bool cmp(const P& a, const P& b) { return a.w
110 < b.w; }
111 LL query(VP& p, int d, int l, int r) {
112     l = lower_bound(p.begin(), p.end(), P{l,
113     -1}, cmp) - p.begin();
114     r = upper_bound(p.begin(), p.end(), P{r,
115     -1}, cmp) - p.begin() - 1;
116     return p[r].s - p[l - 1].s + 1LL * (r - l
117     + 1) * d;
118 }
119 int main() {
120     cin >> n >> Q >> A;
121     FOR (i, 1, n + 1) scanf("%d", &w[i]);
122     FOR (_, 1, n) {
123         int u, v, d; scanf("%d%d%d", &u, &v,
124         &d);
125         G[u].push_back({v, d}); G[v].
126         push_back({u, d});
127     }
128     dfs(1);
129     FOR (i, 1, n + 1)

```

```

128     for (R& x: tr[i]) {
129         x.rt->push_back({w[i], x.dep});
130         if (x.rt2) x.rt2->push_back({w[i]
131         }, x.dep});
132     }
133     FOR (it, pool, pit) {
134         it->push_back({-INF, 0});
135         sort(it->begin(), it->end(), cmp);
136         FOR (i, 1, it->size())
137             (*it)[i].s += (*it)[i - 1].s;
138     }
139     while (Q--) {
140         int u; LL a, b; scanf("%d%d%d", &
141         u, &a, &b);
142         a = (a + ans) % A; b = (b + ans) % A;
143         int l = min(a, b), r = max(a, b);
144         ans = 0;
145         for (R& x: tr[u]) {
146             ans += query(*x.rt, x.dep, l, r
147             );
148             if (x.rt2) ans -= query(*x.rt2,
149             x.dep, l, r);
150         }
151         printf("%lld\n", ans);
152     }
153 }

```

4.6 Heavy-light Decomposition

```

1  // clear clk
2  // usage: hld::predfs(1, 1); hld::dfs(1, 1);
3  int fa[N], dep[N], idx[N], out[N], ridx[N];
4  namespace hld {
5      int sz[N], son[N], top[N], clk;
6      void predfs(int u, int d) {
7          dep[u] = d; sz[u] = 1;
8          int& maxs = son[u] = -1;
9          for (int& v: G[u]) {
10             if (v == fa[u]) continue;
11             fa[v] = u;
12             predfs(v, d + 1);
13             sz[u] += sz[v];
14             if (maxs == -1 || sz[v] > sz[maxs]
15             ) maxs = v;
16         }
17     }
18     void dfs(int u, int tp) {
19         top[u] = tp; idx[u] = ++clk; ridx[clk
20         ] = u;
21         if (son[u] != -1) dfs(son[u], tp);
22         for (int& v: G[u])
23             if (v != fa[u] && v != son[u])
24                 dfs(v, v);
25         out[u] = clk;
26     }
27     template<typename T>
28     int go(int u, int v, T&& f = [](int, int)
29     {
30         int uu = top[u], vv = top[v];
31         while (uu != vv) {
32             if (dep[uu] < dep[vv]) { swap(uu,
33             vv); swap(u, v); }
34             f(idx[uu], idx[u]);
35         }
36     }

```

```

30         u = fa[uu]; uu = top[u];
31     }
32     if (dep[u] < dep[v]) swap(u, v);
33     // choose one
34     // f(idx[v], idx[u]);
35     // if (u != v) f(idx[v] + 1, idx[u]);
36     return v;
37 }
38 int up(int u, int d) {
39     while (d) {
40         if (dep[u] - dep[top[u]] < d) {
41             d -= dep[u] - dep[top[u]];
42             u = top[u];
43         } else return ridx[idx[u] - d];
44         u = fa[u]; --d;
45     }
46     return u;
47 }
48 int finds(int u, int rt) { // find u in
49     // which sub-tree of rt
50     while (top[u] != top[rt]) {
51         u = top[u];
52         if (fa[u] == rt) return u;
53         u = fa[u];
54     }
55     return ridx[idx[rt] + 1];
56 }

```

4.7 Bipartite Matching

```

1 struct MaxMatch {
2     int n;
3     vector<int> G[maxn];
4     int vis[maxn], left[maxn], clk;
5
6     void init(int n) {
7         this->n = n;
8         FOR (i, 0, n + 1) G[i].clear();
9         memset(left, -1, sizeof left);
10        memset(vis, -1, sizeof vis);
11    }
12
13    bool dfs(int u) {
14        for (int v: G[u])
15            if (vis[v] != clk) {
16                vis[v] = clk;
17                if (left[v] == -1 || dfs(left[v])) {
18                    left[v] = u;
19                    return true;
20                }
21            }
22        return false;
23    }
24
25    int match() {
26        int ret = 0;
27        for (clk = 0; clk <= n; ++clk)
28            if (dfs(clk)) ++ret;
29        return ret;
30    }
31 } MM;
32

```

```

33 ///////////////////////////////////////////////////
34 // max weight: KM
35 ///////////////////////////////////////////////////
36 namespace R {
37     const int maxn = 300 + 10;
38     int n, m;
39     int left[maxn], L[maxn], R[maxn];
40     int w[maxn][maxn], slack[maxn];
41     bool visL[maxn], visR[maxn];
42
43     bool dfs(int u) {
44         visL[u] = true;
45         FOR (v, 0, m) {
46             if (visR[v]) continue;
47             int t = L[u] + R[v] - w[u][v];
48             if (t == 0) {
49                 visR[v] = true;
50                 if (left[v] == -1 || dfs(left[v])) {
51                     left[v] = u;
52                     return true;
53                 }
54             } else slack[v] = min(slack[v], t);
55         }
56         return false;
57     }
58
59     int go() {
60         memset(left, -1, sizeof left);
61         memset(R, 0, sizeof R);
62         memset(L, 0, sizeof L);
63         FOR (i, 0, n)
64             FOR (j, 0, m)
65                 L[i] = max(L[i], w[i][j]);
66
67         FOR (i, 0, n) {
68             memset(slack, 0x3f, sizeof slack);
69             while (1) {
70                 memset(visL, 0, sizeof visL);
71                 memset(visR, 0, sizeof visR);
72                 if (dfs(i)) break;
73                 int d = 0x3f3f3f3f;
74                 FOR (j, 0, m) if (!visR[j]) d = min(d, slack[j]);
75                 FOR (j, 0, n) if (visL[j]) L[j] -= d;
76                 FOR (j, 0, m) if (visR[j]) R[j] += d; else slack[j] -= d;
77             }
78             int ret = 0;
79             FOR (i, 0, m) if (left[i] != -1) ret += w[left[i]][i];
80             return ret;
81         }
82     }
83 }

```

4.8 Virtual Tree

```

1 void go(vector<int>& V, int& k) {

```

```

2     int u = V[k]; f[u] = 0;
3     dbg(u, k);
4     for (auto& e: G[u]) {
5         int v = e.to;
6         if (v == pa[u][0]) continue;
7         while (k + 1 < V.size()) {
8             int to = V[k + 1];
9             if (in[to] <= out[v]) {
10                go(V, ++k);
11                if (key[to]) f[u] += w[to];
12                else f[u] += min(f[to], (LL)w[to]);
13            } else break;
14        }
15        dbg(u, f[u]);
16    }
17
18    inline bool cmp(int a, int b) { return in[a] < in[b]; }
19    LL solve(vector<int>& V) {
20        static vector<int> a; a.clear();
21        for (int& x: V) a.push_back(x);
22        sort(a.begin(), a.end(), cmp);
23        FOR (i, 1, a.size())
24            a.push_back(lca(a[i], a[i - 1]));
25        a.push_back(1);
26        sort(a.begin(), a.end(), cmp);
27        a.erase(unique(a.begin(), a.end()), a.end());
28        dbg(a);
29        int tmp; go(a, tmp = 0);
30        return f[1];
31    }

```

4.9 Euler Tour

```

1 int S[N << 1], top;
2 Edge edges[N << 1];
3 set<int> G[N];
4
5 void DFS(int u) {
6     S[top++] = u;
7     for (int eid: G[u]) {
8         int v = edges[eid].get_other(u);
9         G[u].erase(eid);
10        G[v].erase(eid);
11        DFS(v);
12        return;
13    }
14 }
15 void fleury(int start) {
16     int u = start;
17     top = 0; path.clear();
18     S[top++] = u;
19     while (top) {
20         u = S[--top];
21         if (!G[u].empty())
22             DFS(u);
23         else path.push_back(u);
24     }
25 }

```

4.10 SCC, 2-SAT

```

1 int n, m;
2 vector<int> G[N], rG[N], vs;
3 int used[N], cmp[N];
4
5 void add_edge(int from, int to) {
6     G[from].push_back(to);
7     rG[to].push_back(from);
8 }
9
10 void dfs(int v) {
11     used[v] = true;
12     for (int u: G[v]) {
13         if (!used[u])
14             dfs(u);
15     }
16     vs.push_back(v);
17 }
18
19 void rdfs(int v, int k) {
20     used[v] = true;
21     cmp[v] = k;
22     for (int u: rG[v])
23         if (!used[u])
24             rdfs(u, k);
25 }
26
27 int scc() {
28     memset(used, 0, sizeof(used));
29     vs.clear();
30     for (int v = 0; v < n; ++v)
31         if (!used[v]) dfs(v);
32     memset(used, 0, sizeof(used));
33     int k = 0;
34     for (int i = (int) vs.size() - 1; i >= 0; --i)
35         if (!used[vs[i]]) rdfs(vs[i], k++);
36     return k;
37 }
38
39 int main() {
40     cin >> n >> m;
41     n *= 2;
42     for (int i = 0; i < m; ++i) {
43         int a, b; cin >> a >> b;
44         add_edge(a - 1, (b - 1) ^ 1);
45         add_edge(b - 1, (a - 1) ^ 1);
46     }
47     scc();
48     for (int i = 0; i < n; i += 2) {
49         if (cmp[i] == cmp[i + 1]) {
50             puts("NIE");
51             return 0;
52         }
53     }
54     for (int i = 0; i < n; i += 2) {
55         if (cmp[i] > cmp[i + 1]) printf("%d\n", i + 1);
56         else printf("%d\n", i + 2);
57     }
58 }

```

4.11 Topological Sort

```

1 vector<int> toporder(int n) {
2     vector<int> orders;
3     queue<int> q;
4     for (int i = 0; i < n; i++)
5         if (!deg[i]) {
6             q.push(i);
7             orders.push_back(i);
8         }
9     while (!q.empty()) {
10         int u = q.front(); q.pop();
11         for (int v: G[u])
12             if (--deg[v] == 0) {
13                 q.push(v);
14                 orders.push_back(v);
15             }
16     }
17     return orders;
18 }

```

4.12 General Matching

```

1 // O(n^3)
2 vector<int> G[N];
3 int fa[N], mt[N], pre[N], mk[N];
4 int lca_clk, lca_mk[N];
5 pair<int, int> ce[N];
6 void connect(int u, int v) {
7     mt[u] = v;
8     mt[v] = u;
9 }
10 int find(int x) { return x == fa[x] ? x : fa[x] = find(fa[x]); }
11 void flip(int s, int u) {
12     if (s == u) return;
13     if (mk[u] == 2) {
14         int v1 = ce[u].first, v2 = ce[u].second;
15         flip(mt[u], v1);
16         flip(s, v2);
17         connect(v1, v2);
18     } else {
19         flip(s, pre[mt[u]]);
20         connect(pre[mt[u]], mt[u]);
21     }
22 }
23 int get_lca(int u, int v) {
24     lca_clk++;
25     for (u = find(u), v = find(v); ; u = find(pre[u]), v = find(pre[v])) {
26         if (u && lca_mk[u] == lca_clk) return u;
27         lca_mk[u] = lca_clk;
28         if (v && lca_mk[v] == lca_clk) return v;
29         lca_mk[v] = lca_clk;
30     }
31 }
32 void access(int u, int p, const pair<int, int> &c, vector<int> &q) {
33     for (u = find(u); u != p; u = find(pre[u])) {
34         if (mk[u] == 2) {

```

```

35         ce[u] = c;
36         q.push_back(u);
37     }
38     fa[find(u)] = find(p);
39 }
40
41 bool aug(int s) {
42     fill(mk, mk + n + 1, 0);
43     fill(pre, pre + n + 1, 0);
44     iota(fa, fa + n + 1, 0);
45     vector<int> q = {s};
46     mk[s] = 1;
47     int t = 0;
48     for (int t = 0; t < (int) q.size(); ++t) {
49         // q size can be changed
50         int u = q[t];
51         for (int &v: G[u]) {
52             if (find(v) == find(u)) continue;
53             if (!mk[v] && !mt[v]) {
54                 flip(s, u);
55                 connect(u, v);
56                 return true;
57             } else if (!mk[v]) {
58                 int w = mt[v];
59                 mk[v] = 2; mk[w] = 1;
60                 pre[w] = v; pre[v] = u;
61                 q.push_back(w);
62             } else if (mk[find(v)] == 1) {
63                 int p = get_lca(u, v);
64                 access(u, p, {u, v}, q);
65                 access(v, p, {v, u}, q);
66             }
67         }
68     }
69     return false;
70 }
71
72 int match() {
73     fill(mt + 1, mt + n + 1, 0);
74     lca_clk = 0;
75     int ans = 0;
76     FOR(i, 1, n + 1)
77         if (!mt[i]) ans += aug(i);
78     return ans;
79 }

```

4.13 Tarjan

```

1 // articulation points
2 // note that the graph might be disconnected
3 int dfn[N], low[N], clk;
4 void init() { clk = 0; memset(dfn, 0, sizeof(dfn)); }
5 void tarjan(int u, int fa) {
6     low[u] = dfn[u] = ++clk;
7     int cc = fa != -1;
8     for (int &v: G[u]) {
9         if (v == fa) continue;
10        if (!dfn[v]) {
11            tarjan(v, u);
12            low[u] = min(low[u], low[v]);
13            cc += low[v] >= dfn[u];
14        } else low[u] = min(low[u], dfn[v]);

```

```

15     }
16     if (cc > 1) // ...
17 }
18
19 // bridge
20 // note that the graph might have multiple
    edges or be disconnected
21 int dfn[N], low[N], clk;
22 void init() { memset(dfn, 0, sizeof dfn); clk
    = 0; }
23 void tarjan(int u, int fa) {
24     low[u] = dfn[u] = ++clk;
25     int _fst = 0;
26     for (E& e: G[u]) {
27         int v = e.to; if (v == fa && ++_fst
            == 1) continue;
28         if (!dfn[v]) {
29             tarjan(v, u);
30             if (low[v] > dfn[u]) // ...
31                 low[u] = min(low[u], low[v]);
32             else low[u] = min(low[u], dfn[v]);
33         }
34     }
35
36 // scc
37 int low[N], dfn[N], clk, B, bl[N];
38 vector<int> bcc[N];
39 void init() { B = clk = 0; memset(dfn, 0,
    sizeof dfn); }
40 void tarjan(int u) {
41     static int st[N], p;
42     static bool in[N];
43     dfn[u] = low[u] = ++clk;
44     st[p++] = u; in[u] = true;
45     for (int& v: G[u]) {
46         if (!dfn[v]) {
47             tarjan(v);
48             low[u] = min(low[u], low[v]);
49             } else if (in[v]) low[u] = min(low[u]
                , dfn[v]);
50     }
51     if (dfn[u] == low[u]) {
52         while (1) {
53             int x = st[--p]; in[x] = false;
54             bl[x] = B; bcc[B].push_back(x);
55             if (x == u) break;
56         }
57         ++B;
58     }
59 }

```

4.14 Bi-connected Components, Block-cut Tree

```

1 // Array size should be 2 * N
2 // Single edge also counts as bi-connected
    comp
3 // Use |V| <= |E| to filter
4 struct E { int to, nxt; } e[N];
5 int hd[N], ecnt;
6 void addedge(int u, int v) {
7     e[ecnt] = {v, hd[u]};
8     hd[u] = ecnt++;

```

```

9 }
10 int low[N], dfn[N], clk, B, bno[N];
11 vector<int> bc[N], be[N];
12 bool vise[N];
13 void init() {
14     memset(vise, 0, sizeof vise);
15     memset(hd, -1, sizeof hd);
16     memset(dfn, 0, sizeof dfn);
17     memset(bno, -1, sizeof bno);
18     B = clk = ecnt = 0;
19 }
20
21 void tarjan(int u, int feid) {
22     static int st[N], p;
23     static auto add = [&](int x) {
24         if (bno[x] != B) { bno[x] = B; bc[B].
            push_back(x); }
25     };
26     low[u] = dfn[u] = ++clk;
27     for (int i = hd[u]; ~i; i = e[i].nxt) {
28         if ((feid ^ i) == 1) continue;
29         if (!vise[i]) { st[p++] = i; vise[i]
            = vise[i ^ 1] = true; }
30         int v = e[i].to;
31         if (!dfn[v]) {
32             tarjan(v, i);
33             low[u] = min(low[u], low[v]);
34             if (low[v] >= dfn[u]) {
35                 bc[B].clear(); be[B].clear();
36                 while (1) {
37                     int eid = st[--p];
38                     add(e[eid].to); add(e[eid
                        ^ 1].to);
39                     be[B].push_back(eid);
40                     if ((eid ^ i) <= 1) break;
41                 }
42                 ++B;
43             } else low[u] = min(low[u], dfn[v]);
44         }
45     }
46 }
47
48 ///////////////////////////////////////////////////
49 // block-cut tree
50 // cactus -> block-cut tree
51 // N >= |E| * 2
52 ///////////////////////////////////////////////////
53
54 vector<int> G[N];
55 int nn;
56
57 struct E { int to, nxt; };
58 namespace C {
59     E e[N * 2];
60     int hd[N], ecnt;
61     void addedge(int u, int v) {
62         e[ecnt] = {v, hd[u]};
63         hd[u] = ecnt++;
64     }
65
66     int idx[N], clk, fa[N];
67     bool ring[N];
68     void init() { ecnt = 0; memset(hd, -1,
        sizeof hd); clk = 0; }
69     void dfs(int u, int feid) {

```

```

70     idx[u] = ++clk;
71     for (int i = hd[u]; ~i; i = e[i].nxt)
72     {
73         if ((i ^ feid) == 1) continue;
74         int v = e[i].to;
75         if (!idx[v]) {
76             fa[v] = u; ring[u] = false;
77             dfs(v, i);
78             if (!ring[u]) { G[u].
                push_back(v); G[v].
                push_back(u); }
79         } else if (idx[v] < idx[u]) {
80             ++nn;
81             G[nn].push_back(v); G[v].
                push_back(nn); // put the
                root of the cycle in the
                front
82             for (int x = u; x != v; x =
                fa[x]) {
83                 ring[x] = true;
84                 G[nn].push_back(x); G[x].
                push_back(nn);
85             }
86             ring[v] = true;
87         }
88     }
89 }

```

4.15 Minimum Directed Spanning Tree

```

1 // edges will be modified
2 vector<E> edges;
3 int in[N], id[N], pre[N], vis[N];
4 // a copy of n is needed
5 LL zl_tree(int rt, int n) {
6     LL ans = 0;
7     int v, _n = n;
8     while (1) {
9         fill(in, in + n, INF);
10        for (E& e: edges) {
11            if (e.u != e.v && e.w < in[e.v])
12            {
13                pre[e.v] = e.u;
14                in[e.v] = e.w;
15            }
16        }
17        FOR (i, 0, n) if (i != rt && in[i] ==
            INF) return -1;
18        int tn = 0;
19        fill(id, id + _n, -1); fill(vis, vis
            + _n, -1);
20        in[rt] = 0;
21        FOR (i, 0, n) {
22            ans += in[v = i];
23            while (vis[v] != i && id[v] == -1
                && v != rt) {
24                vis[v] = i; v = pre[v];
25            }
26            if (v != rt && id[v] == -1) {
27                for (int u = pre[v]; u != v;
                    u = pre[u]) id[u] = tn;
28                id[v] = tn++;
29            }
30        }
31    }

```

```

29     }
30     if (tn == 0) break;
31     FOR (i, 0, n) if (id[i] == -1) id[i]
32         = tn++;
33     for (int i = 0; i < (int) edges.size
34         (); ) {
35         auto &e = edges[i];
36         v = e.v;
37         e.u = id[e.u]; e.v = id[e.v];
38         if (e.u != e.v) { e.w -= in[v]; i
39             ++; }
40         else { swap(e, edges.back());
41             edges.pop_back(); }
42     }
43     n = tn; rt = id[rt];
44     return ans;
45 }

```

4.16 Cycles

```

1 // refer to cheatsheet for elaboration
2 LL cycle4() {
3     LL ans = 0;
4     iota(kth, kth + n + 1, 0);
5     sort(kth, kth + n, [&](int x, int y) {
6         return deg[x] < deg[y]; });
7     FOR (i, 1, n + 1) rk[kth[i]] = i;
8     FOR (u, 1, n + 1)
9         for (int v: G[u])
10             if (rk[v] > rk[u]) key[u].
11                 push_back(v);
12     FOR (u, 1, n + 1) {
13         for (int v: G[u])
14             for (int w: key[v])
15                 if (rk[w] > rk[u]) ans += cnt
16                     [w]++;
17         for (int v: G[u])
18             for (int w: key[v])
19                 if (rk[w] > rk[u]) --cnt[w];
20     }
21     return ans;
22 }
23 int cycle3() {
24     int ans = 0;
25     for (E &e: edges) { deg[e.u]++; deg[e.v]
26         ++; }
27     for (E &e: edges) {
28         if (deg[e.u] < deg[e.v] || (deg[e.u]
29             == deg[e.v] && e.u < e.v))
30             G[e.u].push_back(e.v);
31         else G[e.v].push_back(e.u);
32     }
33     FOR (x, 1, n + 1) {
34         for (int y: G[x]) p[y] = x;
35         for (int y: G[x]) for (int z: G[y])
36             if (p[z] == x) ans++;
37     }
38     return ans;
39 }

```

4.17 Dominator Tree

```

1 vector<int> G[N], rG[N];
2 vector<int> dt[N];
3
4 namespace tl {
5     int fa[N], idx[N], clk, ridx[N];
6     int c[N], best[N], semi[N], idom[N];
7     void init(int n) {
8         clk = 0;
9         fill(c, c + n + 1, -1);
10        FOR (i, 1, n + 1) dt[i].clear();
11        FOR (i, 1, n + 1) semi[i] = best[i] =
12            i;
13        fill(idx, idx + n + 1, 0);
14    }
15    void dfs(int u) {
16        idx[u] = ++clk; ridx[clk] = u;
17        for (int& v: G[u]) if (!idx[v]) { fa[
18            v] = u; dfs(v); }
19    }
20    int fix(int x) {
21        if (c[x] == -1) return x;
22        int &f = c[x], rt = fix(f);
23        if (idx[semi[best[x]]] > idx[semi[
24            best[f]]]) best[x] = best[f];
25        return f = rt;
26    }
27    void go(int rt) {
28        dfs(rt);
29        FORD (i, clk, 1) {
30            int x = ridx[i], mn = clk + 1;
31            for (int& u: rG[x]) {
32                if (!idx[u]) continue; //
33                reaching all might not be
34                possible
35                fix(u); mn = min(mn, idx[semi
36                    [best[u]]]);
37            }
38            c[x] = fa[x];
39            dt[semi[x]] = ridx[mn].push_back(
40                x);
41            x = ridx[i - 1];
42            for (int& u: dt[x]) {
43                fix(u);
44                if (semi[best[u]] != x) idom[
45                    u] = best[u];
46                else idom[u] = x;
47            }
48            dt[x].clear();
49        }
50        FOR (i, 2, clk + 1) {
51            int u = ridx[i];
52            if (idom[u] != semi[u]) idom[u] =
53                idom[idom[u]];
54            dt[idom[u]].push_back(u);
55        }
56    }
57 }

```

4.18 Global Minimum Cut

```

2 struct StoerWanger {
3     LL n, vis[N];
4     LL dist[N];
5     LL g[N][N];
6
7     void init(int nn, LL w[N][N]) {
8         n = nn;
9         FOR (i, 1, n + 1) FOR (j, 1, n + 1)
10             g[i][j] = w[i][j];
11         memset(dist, 0, sizeof(dist));
12     }
13
14     LL min_cut_phase(int clk, int &x, int &y)
15     {
16         int t;
17         vis[t = 1] = clk;
18         FOR (i, 1, n + 1) if (vis[i] != clk)
19             dist[i] = g[1][i];
20         FOR (i, 1, n) {
21             x = t; t = 0;
22             FOR (j, 1, n + 1)
23                 if (vis[j] != clk && (!t ||
24                     dist[j] > dist[t]))
25                     t = j;
26             vis[t] = clk;
27             FOR (j, 1, n + 1) if (vis[j] !=
28                 clk)
29                 dist[j] += g[t][j];
30         }
31         y = t;
32         return dist[t];
33     }
34
35     void merge(int x, int y) {
36         if (x > y) swap(x, y);
37         FOR (i, 1, n + 1)
38             if (i != x && i != y) {
39                 g[i][x] += g[i][y];
40                 g[x][i] += g[y][i];
41             }
42         if (y == n) return;
43         FOR (i, 1, n) if (i != y) {
44             swap(g[i][y], g[i][n]);
45             swap(g[y][i], g[n][i]);
46         }
47     }
48
49     LL go() {
50         LL ret = INF;
51         memset(vis, 0, sizeof(vis));
52         for (int i = 1, x, y; n > 1; ++i, --n)
53             {
54                 ret = min(ret, min_cut_phase(i, x,
55                     y));
56                 merge(x, y);
57             }
58         return ret;
59     }
60 } sw;

```

5 Geometry

5.1 2D Basics

```

1 int sgn(LD x) { return fabs(x) < eps ? 0 : (x
2   > 0 ? 1 : -1); }
3 struct L;
4 struct P;
5 typedef P V;
6 struct P {
7   LD x, y;
8   explicit P(LD x = 0, LD y = 0): x(x), y(y) {}
9   explicit P(const L& l);
10 };
11 struct L {
12   P s, t;
13   L() {}
14   L(P s, P t): s(s), t(t) {}
15 };
16 P operator + (const P& a, const P& b) {
17   return P(a.x + b.x, a.y + b.y); }
18 P operator - (const P& a, const P& b) {
19   return P(a.x - b.x, a.y - b.y); }
20 P operator * (const P& a, LD k) { return P(a.x * k, a.y * k); }
21 P operator / (const P& a, LD k) { return P(a.x / k, a.y / k); }
22 inline bool operator < (const P& a, const P& b) {
23   return sgn(a.x - b.x) < 0 || (sgn(a.x - b.x) == 0 && sgn(a.y - b.y) < 0);
24 }
25 bool operator == (const P& a, const P& b) {
26   return !sgn(a.x - b.x) && !sgn(a.y - b.y); }
27 P::P(const L& l) { *this = l.t - l.s; }
28 ostream &operator << (ostream &os, const P &p) {
29   return (os << "(" << p.x << ", " << p.y << ")"); }
30 istream &operator >> (istream &is, P &p) {
31   return (is >> p.x >> p.y); }
32 LD dist(const P& p) { return sqrt(p.x * p.x + p.y * p.y); }
33 LD dot(const V& a, const V& b) { return a.x * b.x + a.y * b.y; }
34 LD det(const V& a, const V& b) { return a.x * b.y - a.y * b.x; }
35 LD cross(const P& s, const P& t, const P& o = P()) { return det(s - o, t - o); }

```

5.2 Polar angle sort

```

1 int quad(P p) {
2   int x = sgn(p.x), y = sgn(p.y);
3   if (x > 0 && y >= 0) return 1;
4   if (x <= 0 && y > 0) return 2;
5   if (x < 0 && y <= 0) return 3;

```

```

6   if (x >= 0 && y < 0) return 4;
7   assert(0);
8 }
9 struct cmp_angle {
10   P p;
11   bool operator () (const P& a, const P& b) {
12     {
13       int qa = quad(a - p), qb = quad(b - p);
14       if (qa != qb) return qa < qb; // compare quad
15       int d = sgn(cross(a, b, p));
16       if (d) return d > 0;
17       return dist(a - p) < dist(b - p);
18     }
19   };

```

5.3 Segments, lines

```

1 bool parallel(const L& a, const L& b) {
2   return !sgn(det(P(a), P(b))); }
3 bool l_eq(const L& a, const L& b) {
4   return parallel(a, b) && parallel(L(a.s, b.t), L(b.s, a.t)); }
5 // counter-clockwise r radius
6 P rotation(const P& p, const LD& r) { return P(p.x * cos(r) - p.y * sin(r), p.x * sin(r) + p.y * cos(r)); }
7 P RotateCCW90(const P& p) { return P(-p.y, p.x); }
8 P RotateCW90(const P& p) { return P(p.y, -p.x); }
9 V normal(const V& v) { return V(-v.y, v.x) / dist(v); }
10 // inclusive: <=0; exclusive: <0
11 bool p_on_seg(const P& p, const L& seg) {
12   P a = seg.s, b = seg.t;
13   return !sgn(det(p - a, b - a)) && sgn(dot(p - a, p - b)) <= 0;
14 }
15 LD dist_to_line(const P& p, const L& l) {
16   return fabs(cross(l.s, l.t, p)) / dist(l); }
17 LD dist_to_seg(const P& p, const L& l) {
18   if (l.s == l.t) return dist(p - l);
19   V vs = p - l.s, vt = p - l.t;
20   if (sgn(dot(l, vs)) < 0) return dist(vs);
21   else if (sgn(dot(l, vt)) > 0) return dist(vt);
22   else return dist_to_line(p, l);
23 }
24 // make sure they have intersection in advance
25 P l_intersection(const L& a, const L& b) {
26   LD s1 = det(P(a), b.s - a.s), s2 = det(P(a), b.t - a.s);
27   return (b.s * s2 - b.t * s1) / (s2 - s1);
28 }
29 LD angle(const V& a, const V& b) {

```

```

34 LD r = asin(fabs(det(a, b)) / dist(a) / dist(b));
35 if (sgn(dot(a, b)) < 0) r = PI - r;
36 return r;
37 }
38 // 1: proper; 2: improper
39 int s_l_cross(const L& seg, const L& line) {
40   int d1 = sgn(cross(line.s, line.t, seg.s));
41   int d2 = sgn(cross(line.s, line.t, seg.t));
42   if ((d1 ^ d2) == -2) return 1; // proper
43   if (d1 == 0 || d2 == 0) return 2;
44   return 0;
45 }
46 // 1: proper; 2: improper
47 int s_cross(const L& a, const L& b, P& p) {
48   int d1 = sgn(cross(a.t, b.s, a.s)), d2 = sgn(cross(a.t, b.t, a.s));
49   int d3 = sgn(cross(b.t, a.s, b.s)), d4 = sgn(cross(b.t, a.t, b.s));
50   if ((d1 ^ d2) == -2 && (d3 ^ d4) == -2) {
51     p = l_intersection(a, b); return 1;
52   }
53   if (!d1 && p_on_seg(b.s, a)) { p = b.s; return 2; }
54   if (!d2 && p_on_seg(b.t, a)) { p = b.t; return 2; }
55   if (!d3 && p_on_seg(a.s, b)) { p = a.s; return 2; }
56   if (!d4 && p_on_seg(a.t, b)) { p = a.t; return 2; }
57   return 0;

```

5.4 Polygons

```

1 typedef vector<P> S;
2 // 0 = outside, 1 = inside, -1 = on border
3 int inside(const S& s, const P& p) {
4   int cnt = 0;
5   FOR (i, 0, s.size()) {
6     P a = s[i], b = s[nxt(i)];
7     if (p_on_seg(p, L(a, b))) return -1;
8     if (sgn(a.y - b.y) <= 0) swap(a, b);
9     if (sgn(p.y - a.y) > 0) continue;
10    if (sgn(p.y - b.y) <= 0) continue;
11    cnt += sgn(cross(b, a, p)) > 0;
12  }
13  return bool(cnt & 1);
14 }
15 // can be negative
16 LD polygon_area(const S& s) {
17   LD ret = 0;
18   FOR (i, 1, (LL)s.size() - 1)
19     ret += cross(s[i], s[i + 1], s[0]);
20   return ret / 2;
21 }
22 // duplicate points are not allowed
23 // s is subject to change
24 const int MAX_N = 1000;
25 S convex_hull(S& s) {
26   assert(s.size() >= 3);

```



```

28 sort(s.begin(), s.end());
29 S ret(MAX_N * 2);
30 int sz = 0;
31 FOR (i, 0, s.size()) {
32     while (sz > 1 && sgn(cross(ret[sz - 1], s[i], ret[sz - 2])) < 0) --sz;
33     ret[sz++] = s[i];
34 }
35 int k = sz;
36 FOR (i, (LL)s.size() - 2, -1) {
37     while (sz > k && sgn(cross(ret[sz - 1], s[i], ret[sz - 2])) < 0) --sz;
38     ret[sz++] = s[i];
39 }
40 ret.resize(sz - (s.size() > 1));
41 return ret;
42 }
43 // centroid
44 P ComputeCentroid(const vector<P> &p) {
45     P c(0, 0);
46     LD scale = 6.0 * polygon_area(p);
47     for (unsigned i = 0; i < p.size(); i++) {
48         unsigned j = (i + 1) % p.size();
49         c = c + (p[i] + p[j]) * (p[i].x * p[j].y - p[j].x * p[i].y);
50     }
51     return c / scale;
52 }
53 // Rotating Calipers, find convex hull first
54 LD rotatingCalipers(vector<P> &qs) {
55     int n = qs.size();
56     if (n == 2)
57         return dist(qs[0] - qs[1]);
58     int i = 0, j = 0;
59     FOR (k, 0, n) {
60         if (!qs[i] < qs[k]) i = k;
61         if (qs[j] < qs[k]) j = k;
62     }
63     LD res = 0;
64     int si = i, sj = j;
65     while (i != sj || j != si) {
66         res = max(res, dist(qs[i] - qs[j]));
67         if (sgn(cross(qs[(i+1)%n] - qs[i], qs[(j+1)%n] - qs[j])) < 0)
68             i = (i + 1) % n;
69         else j = (j + 1) % n;
70     }
71     return res;
72 }

```

5.5 Half-plane intersection

```

1 struct LV {
2     P p, v; LD ang;
3     LV() {}
4     LV(P s, P t): p(s), v(t - s) { ang = atan2(v.y, v.x); }
5 }; //
6 bool operator < (const LV &a, const LV &b) {
7     return a.ang < b.ang; }

```

```

8 bool on_left(const LV &l, const P &p) {
9     return sgn(cross(l.v, p - l.p)) >= 0; }
10 P l_intersection(const LV &a, const LV &b) {
11     P u = a.p - b.p; LD t = cross(b.v, u) / cross(a.v, b.v);
12     return a.p + a.v * t;
13 }
14 S half_plane_intersection(vector<LV> &L) {
15     int n = L.size(), fi, la;
16     sort(L.begin(), L.end());
17     vector<P> p(n); vector<LV> q(n);
18     q[fi = la = 0] = L[0];
19     FOR (i, 1, n) {
20         while (fi < la && !on_left(L[i], p[la - 1])) la--;
21         while (fi < la && !on_left(L[i], p[fi])) fi++;
22         q[++la] = L[i];
23         if (sgn(cross(q[la].v, q[la - 1].v)) == 0) {
24             la--;
25             if (on_left(q[la], L[i].p)) q[la] = L[i];
26         }
27         if (fi < la) p[la - 1] = l_intersection(q[la - 1], q[la]);
28     }
29     while (fi < la && !on_left(q[fi], p[la - 1])) la--;
30     if (la - fi <= 1) return vector<P>();
31     p[la] = l_intersection(q[la], q[fi]);
32     return vector<P>(p.begin() + fi, p.begin() + la + 1);
33 }
34 S convex_intersection(const vector<P> &v1, const vector<P> &v2) {
35     vector<LV> h; int n = v1.size(), m = v2.size();
36     FOR (i, 0, n) h.push_back(LV(v1[i], v1[(i + 1) % n]));
37     FOR (i, 0, m) h.push_back(LV(v2[i], v2[(i + 1) % m]));
38     return half_plane_intersection(h);
39 }

```

5.6 Circles

```

1 struct C {
2     P p; LD r;
3     C(LD x = 0, LD y = 0, LD r = 0): p(x, y), r(r) {}
4     C(P p, LD r): p(p), r(r) {}
5 };
6 P compute_circle_center(P a, P b, P c) {
7     b = (a + b) / 2;
8     c = (a + c) / 2;
9     return l_intersection({b, b + RotateCW90(a - b)}, {c, c + RotateCW90(a - c)});
10 }

```

```

13 // intersections are clockwise subject to center
14 vector<P> c_l_intersection(const L &l, const C &c) {
15     vector<P> ret;
16     P b(l), a = l.s - c.p;
17     LD x = dot(b, b), y = dot(a, b), z = dot(a, a) - c.r * c.r;
18     LD D = y * y - x * z;
19     if (sgn(D) < 0) return ret;
20     ret.push_back(c.p + a + b * (-y + sqrt(D + eps)) / x);
21     if (sgn(D) > 0) ret.push_back(c.p + a + b * (-y - sqrt(D)) / x);
22     return ret;
23 }
24 vector<P> c_c_intersection(C a, C b) {
25     vector<P> ret;
26     LD d = dist(a.p - b.p);
27     if (sgn(d) == 0 || sgn(d - (a.r + b.r)) > 0 || sgn(d + min(a.r, b.r) - max(a.r, b.r)) < 0)
28         return ret;
29     LD x = (d * d - b.r * b.r + a.r * a.r) / (2 * d);
30     LD y = sqrt(a.r * a.r - x * x);
31     P v = (b.p - a.p) / d;
32     ret.push_back(a.p + v * x + RotateCCW90(v) * y);
33     if (sgn(y) > 0) ret.push_back(a.p + v * x - RotateCCW90(v) * y);
34     return ret;
35 }
36 // 1: inside, 2: internally tangent
37 // 3: intersect, 4: ext tangent 5: outside
38 int c_c_relation(const C &a, const C &b) {
39     LD d = dist(a.p - b.p);
40     if (sgn(d - a.r - b.r) > 0) return 5;
41     if (sgn(d - a.r - b.r) == 0) return 4;
42     LD l = fabs(a.r - b.r);
43     if (sgn(d - l) > 0) return 3;
44     if (sgn(d - l) == 0) return 2;
45     if (sgn(d - l) < 0) return 1;
46 }
47 // circle triangle intersection
48 // abs might be needed
49 LD sector_area(const P &a, const P &b, LD r) {
50     LD th = atan2(a.y, a.x) - atan2(b.y, b.x);
51     while (th <= 0) th += 2 * PI;
52     while (th > 2 * PI) th -= 2 * PI;
53     th = min(th, 2 * PI - th);
54     return r * r * th / 2;
55 }
56 LD c_tri_area(P a, P b, P center, LD r) {
57     a = a - center; b = b - center;
58     int ina = sgn(dist(a) - r) < 0, inb = sgn(dist(b) - r) < 0;
59     // dbg(a, b, ina, inb);
60     if (ina && inb) {
61         return fabs(cross(a, b)) / 2;
62     } else {
63         return 0;
64     }
65 }

```

```

65 auto p = c_l_intersection(L(a, b), C
66   (0, 0, r));
67 if (ina ^ inb) {
68   auto cr = p_on_seg(p[0], L(a, b))
69     ? p[0] : p[1];
70   if (ina) return sector_area(b, cr
71     , r) + fabs(cross(a, cr)) /
72     2;
73   else return sector_area(a, cr, r)
74     + fabs(cross(b, cr)) / 2;
75 } else {
76   if ((int) p.size() == 2 &&
77     p_on_seg(p[0], L(a, b))) {
78     if (dist(p[0] - a) > dist(p
79       [1] - a)) swap(p[0], p
80       [1]);
81     return sector_area(a, p[0], r
82       ) + sector_area(p[1], b,
83       r)
84     + fabs(cross(p[0], p[1]))
85     / 2;
86   } else return sector_area(a, b, r
87     );
88 }
89 }
90 typedef vector<P> S;
91 LD c_poly_area(S poly, const C& c) {
92   LD ret = 0; int n = poly.size();
93   FOR (i, 0, n) {
94     int t = sgn(cross(poly[i] - c.p, poly
95       [(i + 1) % n] - c.p));
96     if (t) ret += t * c_tri_area(poly[i],
97       poly[(i + 1) % n], c.p, c.r);
98   }
99   return ret;
100 }

```

5.7 Circle Union

```

1 // version 1
2 // union O(n^3 log n)
3 struct CV {
4   LD yl, yr, ym; C o; int type;
5   CV() {}
6   CV(LD yl, LD yr, LD ym, C c, int t)
7     : yl(yl), yr(yr), ym(ym), type(t), o(
8       c) {}
9 };
10 pair<LD, LD> c_point_eval(const C& c, LD x) {
11   LD d = fabs(c.p.x - x), h = rt(sq(c.r) -
12     sq(d));
13   return {c.p.y - h, c.p.y + h};
14 }
15 pair<CV, CV> pairwise_curves(const C& c, LD
16   xl, LD xr) {
17   LD yl1, yl2, yr1, yr2, ym1, ym2;
18   tie(yl1, yl2) = c_point_eval(c, xl);
19   tie(ym1, ym2) = c_point_eval(c, (xl + xr)
20     / 2);
21   tie(yr1, yr2) = c_point_eval(c, xr);
22   return {CV(yl1, yr1, ym1, c, 1), CV(yl2,
23     yr2, ym2, c, -1)};
24 }

```

```

25 bool operator < (const CV& a, const CV& b) {
26   return a.ym < b.ym; }
27 LD cv_area(const CV& v, LD xl, LD xr) {
28   LD l = rt(sq(xr - xl) + sq(v.yr - v.yl));
29   LD d = rt(sq(v.o.r) - sq(l / 2));
30   LD ang = atan(l / d / 2);
31   return ang * sq(v.o.r) - d * l / 2;
32 }
33 LD circle_union(const vector<C>& cs) {
34   int n = cs.size();
35   vector<LD> xs;
36   FOR (i, 0, n) {
37     xs.push_back(cs[i].p.x - cs[i].r);
38     xs.push_back(cs[i].p.x);
39     xs.push_back(cs[i].p.x + cs[i].r);
40   }
41   FOR (j, i + 1, n) {
42     auto pts = c_c_intersection(cs[i]
43       , cs[j]);
44     for (auto& p: pts) xs.push_back(p
45       .x);
46   }
47   sort(xs.begin(), xs.end());
48   xs.erase(unique(xs.begin(), xs.end(), [] (
49     LD x, LD y) { return sgn(x - y) == 0;
50     }), xs.end());
51   LD ans = 0;
52   FOR (i, 0, (int) xs.size() - 1) {
53     LD xl = xs[i], xr = xs[i + 1];
54     vector<CV> intv;
55     FOR (k, 0, n) {
56       auto& c = cs[k];
57       if (sgn(c.p.x - c.r - xl) <= 0 &&
58         sgn(c.p.x + c.r - xr) >= 0) {
59         auto t = pairwise_curves(c,
60           xl, xr);
61         intv.push_back(t.first); intv
62           .push_back(t.second);
63       }
64     }
65     sort(intv.begin(), intv.end());
66     vector<LD> areas(intv.size());
67     FOR (i, 0, intv.size()) areas[i] =
68       cv_area(intv[i], xl, xr);
69     int cc = 0;
70     FOR (i, 0, intv.size()) {
71       if (cc > 0) {
72         ans += (intv[i].yl - intv[i -
73           1].yl + intv[i].yr -
74           intv[i - 1].yr) * (xr -
75           xl) / 2;
76         ans += intv[i].type *
77           areas[i - 1];
78         ans -= intv[i].type * areas[i]
79           ;
80       }
81       cc += intv[i].type;
82     }
83     return ans;
84 }
85 // version 2 (k-cover, O(n^2 log n))

```

```

71 inline LD angle(const P &p) { return atan2(p.
72   y, p.x); }
73 // Points on circle
74 // p is coordinates relative to c
75 struct CP {
76   P p;
77   LD a;
78   int t;
79   CP() {}
80   CP(P p, LD a, int t) : p(p), a(a), t(t) {}
81 };
82 bool operator<(const CP &u, const CP &v) {
83   return u.a < v.a; }
84 LD cv_area(LD r, const CP &q1, const CP &q2)
85 {
86   return (r * r * (q2.a - q1.a) - cross(q1.p,
87     q2.p)) / 2;
88 }
89 LD ans[N];
90 void circle_union(const vector<C> &cs) {
91   int n = cs.size();
92   FOR (i, 0, n) {
93     // same circle, only the first one counts
94     bool ok = true;
95     FOR (j, 0, i)
96       if (sgn(cs[i].r - cs[j].r) == 0 && cs[i].
97         p == cs[j].p) {
98         ok = false;
99         break;
100       }
101     if (!ok) continue;
102     auto &c = cs[i];
103     vector<CP> ev;
104     int belong_to = 0;
105     P bound = c.p + P(-c.r, 0);
106     ev.emplace_back(bound, -PI, 0);
107     ev.emplace_back(bound, PI, 0);
108     FOR (j, 0, n) {
109       if (i == j) continue;
110       if (c_c_relation(c, cs[j]) <= 2) {
111         if (sgn(cs[j].r - c.r) >= 0) //
112           totally covered
113           belong_to++;
114         continue;
115       }
116       auto its = c_c_intersection(c, cs[j]);
117       if (its.size() == 2) {
118         P p = its[1] - c.p, q = its[0] - c.p;
119         LD a = angle(p), b = angle(q);
120         if (sgn(a - b) > 0) {
121           ev.emplace_back(p, a, 1);
122           ev.emplace_back(bound, PI, -1);
123           ev.emplace_back(bound, -PI, 1);
124           ev.emplace_back(q, b, -1);
125         } else {
126           ev.emplace_back(p, a, 1);
127           ev.emplace_back(q, b, -1);
128         }
129       }
130     }
131     sort(ev.begin(), ev.end());
132     int cc = ev[0].t;

```

```

131 FOR(j, 1, ev.size()) {
132     int t = cc + belong_to;
133     ans[t] += cross(ev[j - 1].p + c.p, ev[j]
134                   ].p + c.p) / 2;
135     ans[t] += cv_area(c.r, ev[j - 1], ev[j]
136                   );
137     cc += ev[j].t;
138 }

```

5.8 Minimum Covering Circle

```

1 P compute_circle_center(P a, P b) { return (a
2   + b) / 2; }
3 bool p_in_circle(const P& p, const C& c) {
4   return sgn(dist(p - c.p) - c.r) <= 0;
5 }
6 C min_circle_cover(const vector<P> &in) {
7   vector<P> a(in.begin(), in.end());
8   dbg(a.size());
9   random_shuffle(a.begin(), a.end());
10  P c = a[0]; LD r = 0; int n = a.size();
11  FOR(i, 1, n) if (!p_in_circle(a[i], {c,
12    r})) {
13    c = a[i]; r = 0;
14    FOR(j, 0, i) if (!p_in_circle(a[j],
15      {c, r})) {
16      c = compute_circle_center(a[i], a
17        [j]);
18      r = dist(a[j] - c);
19      FOR(k, 0, j) if (!p_in_circle(a[
20        k], {c, r})) {
21        c = compute_circle_center(a[i
22          ], a[j], a[k]);
23        r = dist(a[k] - c);
24      }
25    }
26  }
27  return {c, r};
28 }

```

5.9 Circle Inversion

```

1 C inv(C c, const P& o) {
2   LD d = dist(c.p - o);
3   assert(sgn(d) != 0);
4   LD a = 1 / (d - c.r);
5   LD b = 1 / (d + c.r);
6   c.r = (a - b) / 2 * R2;
7   c.p = o + (c.p - o) * ((a + b) * R2 / 2 /
8     d);
9   return c;
10 }

```

5.10 3D Basics

```

1 struct P;

```

```

2 struct L;
3 typedef P V;
4 struct P {
5   LD x, y, z;
6   explicit P(LD x = 0, LD y = 0, LD z = 0):
7     x(x), y(y), z(z) {}
8   explicit P(const L& l);
9 };
10 struct L {
11   P s, t;
12   L() {}
13   L(P s, P t): s(s), t(t) {}
14 };
15 struct F {
16   P a, b, c;
17   F() {}
18   F(P a, P b, P c): a(a), b(b), c(c) {}
19 };
20 P operator + (const P& a, const P& b) {}
21 P operator - (const P& a, const P& b) {}
22 P operator * (const P& a, LD k) {}
23 P operator / (const P& a, LD k) {}
24 inline int operator < (const P& a, const P& b)
25 {
26   return sgn(a.x - b.x) < 0 || (sgn(a.x - b
27     .x) == 0 && (sgn(a.y - b.y) < 0 ||
28       (sgn(a.y - b.y) == 0 &&
29         sgn(a.z - b.z) < 0)))
30     ;
31 }
32 bool operator == (const P& a, const P& b) {
33   return !sgn(a.x - b.x) && !sgn(a.y - b.y)
34     && !sgn(a.z - b.z);
35 }
36 P::P(const L& l) { *this = l.t - l.s; }
37 ostream &operator << (ostream &os, const P &p
38 ) {
39   return (os << "(" << p.x << ", " << p.y <<
40     ", " << p.z << ")");
41 }
42 istream &operator >> (istream &is, P &p) {
43   return (is >> p.x >> p.y >> p.z);
44 }
45 LD dist2(const P& p) { return p.x * p.x + p.y
46   * p.y + p.z * p.z; }
47 LD dist(const P& p) { return sqrt(dist2(p)); }
48 LD dot(const V& a, const V& b) { return a.x *
49   b.x + a.y * b.y + a.z * b.z; }
50 P cross(const P& v, const P& w) {
51   return P(v.y * w.z - v.z * w.y, v.z * w.x
52     - v.x * w.z, v.x * w.y - v.y * w.x);
53 }
54 LD mix(const V& a, const V& b, const V& c) {
55   return dot(a, cross(b, c));
56 }
57 // counter-clockwise r radius
58 // axis = 0 around axis x
59 // axis = 1 around axis y
60 // axis = 2 around axis z
61 P rotation(const P& p, const LD& r, int axis
62   = 0) {
63   if (axis == 0)

```

```

48   return P(p.x, p.y * cos(r) - p.z *
49     sin(r), p.y * sin(r) + p.z * cos(
50     r));
51   else if (axis == 1)
52     return P(p.z * cos(r) - p.x * sin(r),
53       p.y, p.z * sin(r) + p.x * cos(r)
54       );
55   else if (axis == 2)
56     return P(p.x * cos(r) - p.y * sin(r),
57       p.x * sin(r) + p.y * cos(r), p.z
58       );
59 }
60 // n is normal vector
61 // this is clockwise
62 P rotation(const P& p, const LD& r, const P&
63   n) {
64   LD c = cos(r), s = sin(r), x = n.x, y = n
65     .y, z = n.z;
66   return P((x * x * (1 - c) + c) * p.x + (x
67     * y * (1 - c) + z * s) * p.y + (x *
68     z * (1 - c) - y * s) * p.z,
69     (x * y * (1 - c) - z * s) * p.x
70     + (y * y * (1 - c) + c) * p.y
71     + (y * z * (1 - c) + x * s)
72     ) * p.z,
73     (x * z * (1 - c) + y * s) * p.x
74     + (y * z * (1 - c) - x * s)
75     ) * p.y + (z * z * (1 - c) + c)
76     ) * p.z;
77 }

```

5.11 3D Line, Face

```

1 // <= 0 improper, < 0 proper
2 bool p_on_seg(const P& p, const L& seg) {
3   P a = seg.s, b = seg.t;
4   return !sgn(dist2(cross(p - a, b - a)))
5     && sgn(dot(p - a, p - b)) <= 0;
6 }
7 LD dist_to_line(const P& p, const L& l) {
8   return dist(cross(l.s - p, l.t - p)) /
9     dist(l);
10 }
11 LD dist_to_seg(const P& p, const L& l) {
12   if (l.s == l.t) return dist(p - l.s);
13   V vs = p - l.s, vt = p - l.t;
14   if (sgn(dot(l, vs)) < 0) return dist(vs);
15   else if (sgn(dot(l, vt)) > 0) return dist
16     (vt);
17   else return dist_to_line(p, l);
18 }
19 P norm(const F& f) { return cross(f.a - f.b,
20   f.b - f.c); }
21 int p_on_plane(const F& f, const P& p) {
22   return sgn(dot(norm(f), p - f.a)) == 0; }
23 // if two points are on the opposite side of
24 // a line
25 // return 0 if points is on the line
26 // makes no sense if points and line are not
27 // coplanar
28 int opposite_side(const P& u, const P& v,
29   const L& l) {

```

```

23     return sgn(dot(cross(P(1), u - l.s), cross(
24         P(1), v - l.s))) < 0;
25 }
26 bool parallel(const L& a, const L& b) {
27     return !sgn(dist2(cross(P(a), P(b)))); }
28 int s_intersect(const L& u, const L& v) {
29     return p_on_plane(F(u.s, u.t, v.s), v.t)
30         && opposite_side(u.s, u.t, v) &&
31         opposite_side(v.s, v.t, u);
32 }

```

5.12 3D Convex

```

1 struct FT {
2     int a, b, c;
3     FT() {}
4     FT(int a, int b, int c) : a(a), b(b), c(c) {}
5 };
6
7 bool p_on_line(const P& p, const L& l) {
8     return !sgn(dist2(cross(p - l.s, P(l))));
9 }
10
11 vector<F> convex_hull(vector<P> &p) {
12     sort(p.begin(), p.end());
13     p.erase(unique(p.begin(), p.end(), p.end())
14         ());
15     random_shuffle(p.begin(), p.end());
16     vector<FT> face;
17     FOR (i, 2, p.size()) {
18         if (p_on_line(p[i], L(p[0], p[1])))
19             continue;
20         swap(p[i], p[2]);
21         FOR (j, i + 1, p.size()) {
22             if (sgn(mix(p[1] - p[0], p[2] - p[1],
23                 p[j] - p[0]))) {
24                 swap(p[j], p[3]);
25                 face.emplace_back(0, 1, 2);
26                 face.emplace_back(0, 2, 1);
27                 goto found;
28             }
29         }
30     }
31     found:
32     vector<vector<int>> mk(p.size(), vector<
33         int>(p.size()));
34     FOR (v, 3, p.size()) {
35         vector<FT> tmp;
36         FOR (i, 0, face.size()) {
37             int a = face[i].a, b = face[i].b,
38                 c = face[i].c;
39             if (sgn(mix(p[a] - p[v], p[b] - p[v],
40                 p[c] - p[v])) < 0) {
41                 mk[a][b] = mk[b][a] = v;
42                 mk[b][c] = mk[c][b] = v;
43                 mk[c][a] = mk[a][c] = v;
44             } else tmp.push_back(face[i]);
45         }
46         face = tmp;
47         FOR (i, 0, tmp.size()) {
48             int a = face[i].a, b = face[i].b,
49                 c = face[i].c;

```

```

42         if (mk[a][b] == v) face.
43             emplace_back(b, a, v);
44         if (mk[b][c] == v) face.
45             emplace_back(c, b, v);
46         if (mk[c][a] == v) face.
47             emplace_back(a, c, v);
48     }
49     vector<F> out;
50     FOR (i, 0, face.size())
51         out.emplace_back(p[face[i].a], p[face[i].b], p[face[i].c]);
52     return out;
53 }

```

6 String

6.1 Aho-Corasick Automation

```

1 const int N = 1e6 + 100, M = 26;
2 int mp(char ch) { return ch - 'a'; }
3 struct ACA {
4     int ch[N][M], danger[N], fail[N];
5     int sz;
6     void init() {
7         sz = 1;
8         memset(ch[0], 0, sizeof ch[0]);
9         memset(danger, 0, sizeof danger);
10    }
11    void insert(const string &s, int m) {
12        int n = s.size(); int u = 0, c;
13        FOR (i, 0, n) {
14            c = mp(s[i]);
15            if (!ch[u][c]) {
16                memset(ch[sz], 0, sizeof ch[sz]);
17                danger[sz] = 0; ch[u][c] = sz;
18                ++sz;
19            }
20            u = ch[u][c];
21        }
22        danger[u] |= 1 << m;
23    }
24    void build() {
25        queue<int> Q;
26        fail[0] = 0;
27        for (int c = 0, u; c < M; c++) {
28            u = ch[0][c];
29            if (u) { Q.push(u); fail[u] = 0; }
30        }
31        while (!Q.empty()) {
32            int r = Q.front(); Q.pop();
33            danger[r] |= danger[fail[r]];
34            for (int c = 0, u; c < M; c++) {
35                u = ch[r][c];
36                if (!u) {
37                    ch[r][c] = ch[fail[r]][c];
38                    continue;
39                }
40                fail[u] = ch[fail[r]][c];

```

```

40         Q.push(u);
41     }
42 }
43 }
44 } ac;
45
46 char s[N];
47 int main() {
48     int n; scanf("%d", &n);
49     ac.init();
50     while (n--) {
51         scanf("%s", s);
52         ac.insert(s, 0);
53     }
54     ac.build();
55     scanf("%s", s);
56     int u = 0; n = strlen(s);
57     FOR (i, 0, n) {
58         u = ac.ch[u][mp(s[i])];
59         if (ac.danger[u]) {
60             puts("YES");
61             return 0;
62         }
63     }
64     puts("NO");
65     return 0;
66 }

```

6.2 Hash

```

1 const int p1 = 1e9 + 7, p2 = 1e9 + 9;
2 ULL xp1[N], xp2[N], xp[N];
3 void init_xp() {
4     xp1[0] = xp2[0] = xp[0] = 1;
5     for (int i = 1; i < N; ++i) {
6         xp1[i] = xp1[i - 1] * x % p1;
7         xp2[i] = xp2[i - 1] * x % p2;
8         xp[i] = xp[i - 1] * x;
9     }
10 }
11 struct String {
12     char s[N];
13     int length, subsize;
14     bool sorted;
15     ULL h[N], hl[N];
16     ULL hash() {
17         length = strlen(s);
18         ULL res1 = 0, res2 = 0;
19         h[length] = 0; // ATTENTION!
20         for (int j = length - 1; j >= 0; --j) {
21             #ifdef ENABLE_DOUBLE_HASH
22                 res1 = (res1 * x + s[j]) % p1;
23                 res2 = (res2 * x + s[j]) % p2;
24                 h[j] = (res1 << 32) | res2;
25             #else
26                 res1 = res1 * x + s[j];
27                 h[j] = res1;
28             #endif
29             // printf("%llu\n", h[j]);
30         }
31         return h[0];
32     }
33     // hash of [left, right)

```

```

34 ULL get_substring_hash(int left, int
35 right) const {
36     int len = right - left;
37     #ifndef ENABLE_DOUBLE_HASH
38     // get hash of s[left...right-1]
39     unsigned int mask32 = ~(0u);
40     ULL left1 = h[left] >> 32, right1 = h
41     [right] >> 32;
42     ULL left2 = h[left] & mask32, right2
43     = h[right] & mask32;
44     return (((left1 - right1 * xp1[len] %
45     p1 + p1) % p1) << 32) |
46     (((left2 - right2 * xp2[len] %
47     p2 + p2) % p2));
48 #else
49     return h[left] - h[right] * xp[len];
50 #endif
51 }
52 void get_all_subs_hash(int sublen) {
53     subsize = length - sublen + 1;
54     for (int i = 0; i < subsize; ++i)
55         hl[i] = get_substring_hash(i, i +
56         sublen);
57     sorted = 0;
58 }
59 void sort_substring_hash() {
60     sort(hl, hl + subsize);
61     sorted = 1;
62 }
63 bool match(ULL key) const {
64     if (!sorted) assert(0);
65     if (!subsize) return false;
66     return binary_search(hl, hl + subsize
67     , key);
68 }
69 void init(const char *t) {
70     length = strlen(t);
71     strcpy(s, t);
72 }
73 };
74 int LCP(const String &a, const String &b, int
75 ai, int bi) {
76     // Find LCP of a[ai...] and b[bi...]
77     int l = 0, r = min(a.length - ai, b.
78     length - bi);
79     while (l < r) {
80         int mid = (l + r + 1) / 2;
81         if (a.get_substring_hash(ai, ai + mid
82         ) == b.get_substring_hash(bi, bi
83         + mid))
84             l = mid;
85         else r = mid - 1;
86     }
87     return l;
88 }

```

6.3 KMP

```

1 void get_pi(int a[], char s[], int n) {
2     int j = a[0] = 0;
3     FOR (i, 1, n) {
4         while (j && s[i] != s[j]) j = a[j -
5         1];
6         a[i] = j += s[i] == s[j];

```

```

6     }
7 }
8 void get_z(int a[], char s[], int n) {
9     int l = 0, r = 0; a[0] = n;
10    FOR (i, 1, n) {
11        a[i] = i > r ? 0 : min(r - i + 1, a[i
12        - l]);
13        while (i + a[i] < n && s[a[i]] == s[i
14        + a[i]]) ++a[i];
15        if (i + a[i] - 1 > r) { l = i; r = i
16        + a[i] - 1; }
17    }
18 }

```

6.4 Manacher

```

1 int RL[N];
2 void manacher(int* a, int n) { // "abc" => "#
3     a##a##"
4     int r = 0, p = 0;
5     FOR (i, 0, n) {
6         if (i < r) RL[i] = min(RL[2 * p - i],
7         r - i);
8         else RL[i] = 1;
9         while (i - RL[i] >= 0 && i + RL[i] <
10        n && a[i - RL[i]] == a[i + RL[i]
11        ])
12            RL[i]++;
13        if (RL[i] + i - 1 > r) { r = RL[i] +
14        i - 1; p = i; }
15    }
16    FOR (i, 0, n) --RL[i];

```

6.5 Palindrome Automation

```

1 // num: the number of palindrome suffixes of
2 // the prefix represented by the node
3 // cnt: the number of occurrences in string (
4 // should update to father before using)
5 namespace pam {
6     int t[N][26], fa[N], len[N], rs[N], cnt[N]
7     ], num[N];
8     int sz, n, last;
9     int _new(int l) {
10         memset(t[sz], 0, sizeof t[0]);
11         len[sz] = l; cnt[sz] = num[sz] = 0;
12         return sz++;
13     }
14     void init() {
15         rs[n = sz = 0] = -1;
16         last = _new(0);
17         fa[last] = _new(-1);
18     }
19     int get_fa(int x) {
20         while (rs[n - 1 - len[x]] != rs[n]) x
21         = fa[x];
22         return x;
23     }
24     void ins(int ch) {
25         rs[++n] = ch;

```

```

22     int p = get_fa(last);
23     if (!t[p][ch]) {
24         int np = _new(len[p] + 2);
25         num[np] = num[fa[np]] = t[get_fa(
26         fa[p])[ch] + 1;
27         t[p][ch] = np;
28     }
29     ++cnt[last = t[p][ch]];
30 }

```

6.6 Suffix Array

```

1 struct SuffixArray {
2     const int L;
3     vector<vector<int>> P;
4     vector<pair<pair<int, int>, int>> M;
5     int s[N], sa[N], rank[N], height[N];
6     // s: raw string
7     // sa[i]=k: s[k...L-1] ranks i (0 based)
8     // rank[i]=k: the rank of s[i...L-1] is k
9     // height[i] = lcp(sa[i-1], sa[i])
10    SuffixArray(const string &raw_s) : L(
11    raw_s.length()), P(1, vector<int>(L,
12    0)), M(L) {
13        for (int i = 0; i < L; ++i)
14            P[0][i] = this->s[i] = int(raw_s[
15            i]);
16        for (int skip = 1, level = 1; skip <
17        L; skip *= 2, level++) {
18            P.push_back(vector<int>(L, 0));
19            for (int i = 0; i < L; i++)
20                M[i] = make_pair(make_pair(P[
21                level - 1][i], i + skip <
22                L ? P[level - 1][i +
23                skip] : -1000), i);
24            sort(M.begin(), M.end());
25            for (int i = 0; i < L; i++)
26                P[level][M[i].second] = (i >
27                0 && M[i].first == M[i -
28                1].first) ? P[level][M[i
29                - 1].second] : i;
30        }
31        for (unsigned i = 0; i < P.back().
32        size(); ++i) {
33            rank[i] = P.back()[i];
34            sa[rank[i]] = i;
35        }
36        // This is a traditional way to calculate
37        LCP
38        void getHeight() {
39            memset(height, 0, sizeof height);
40            int k = 0;
41            for (int i = 0; i < L; ++i) {
42                if (rank[i] == 0) continue;
43                if (k) k--;
44                int j = sa[rank[i] - 1];
45                while (i + k < L && j + k < L &&
46                s[i + k] == s[j + k]) ++k;
47                height[rank[i]] = k;
48            }
49            rmq_init(height, L);

```

```

38 }
39 int f[N][Nlog];
40 inline int highbit(int x) {
41     return 31 - __builtin_clz(x);
42 }
43 int rmq_query(int x, int y) {
44     int p = highbit(y - x + 1);
45     return min(f[x][p], f[y - (1 << p) + 1][p]);
46 }
47 // arr has to be 0 based
48 void rmq_init(int *arr, int length) {
49     for (int x = 0; x <= highbit(length); ++x)
50         for (int i = 0; i <= length - (1 << x); ++i) {
51             if (!x) f[i][x] = arr[i];
52             else f[i][x] = min(f[i][x - 1], f[i + (1 << (x - 1))][x - 1]);
53         }
54 }
55 #ifdef NEW
56 // returns the length of the longest common prefix of s[i...L-1] and s[j...L-1]
57 int LongestCommonPrefix(int i, int j) {
58     int len = 0;
59     if (i == j) return L - i;
60     for (int k = (int) P.size() - 1; k >= 0 && i < L && j < L; k--) {
61         if (P[k][i] == P[k][j]) {
62             i += 1 << k;
63             j += 1 << k;
64             len += 1 << k;
65         }
66     }
67     return len;
68 }
69 #else
70 int LongestCommonPrefix(int i, int j) {
71     // getHeight() must be called first
72     if (i == j) return L - i;
73     if (i > j) swap(i, j);
74     return rmq_query(i + 1, j);
75 }
76 #endif
77 int checkNonOverlappingSubstring(int K) {
78     // check if there is two non-overlapping identical substring of length K
79     int minsa = 0, maxsa = 0;
80     for (int i = 0; i < L; ++i) {
81         if (height[i] < K) {
82             minsa = sa[i]; maxsa = sa[i];
83         } else {
84             minsa = min(minsa, sa[i]);
85             maxsa = max(maxsa, sa[i]);
86             if (maxsa - minsa >= K) return 1;
87         }
88     }
89     return 0;
90 }
91 int checkBelongToDifferentSubstring(int K, int split) {
92     int minsa = 0, maxsa = 0;
93     for (int i = 0; i < L; ++i) {
94         if (height[i] < K) {
95             minsa = sa[i]; maxsa = sa[i];
96         } else {
97             minsa = min(minsa, sa[i]);
98             maxsa = max(maxsa, sa[i]);
99             if (maxsa > split && minsa < split) return 1;
100         }
101     }
102     return 0;
103 }
104 } *S;
105 int main() {
106     int sp = s.length();
107     s += "*" + t;
108     S = new SuffixArray(s);
109     S->getHeight();
110     int left = 0, right = sp;
111     while (left < right) {
112         // ...
113         if (S->checkBelongToDifferentSubstring(mid, sp)) // ...
114     }
115     printf("%d\n", left);
116 }
117 ///////////////////////////////////////////////////////////////////
118 // rk [0..n-1] -> [1..n], sa/ht [1..n]
119 // s[i] > 0 && s[n] = 0
120 // b: normally as bucket
121 // c: normally as bucket1
122 // d: normally as bucket2
123 // f: normally as cntbuf
124 ///////////////////////////////////////////////////////////////////
125 template<size_t size>
126 struct SuffixArray {
127     bool t[size << 1];
128     int b[size], c[size];
129     int sa[size], rk[size], ht[size];
130     inline bool isLMS(const int i, const bool *t) { return i > 0 && t[i] && !t[i - 1]; }
131 }
132 template<class T>
133 inline void inducedSort(T s, int *sa, const int n, const int M, const int bs, bool *t, int *b, int *f, int *p) {
134     fill(b, b + M, 0); fill(sa, sa + n, -1);
135     FOR (i, 0, n) b[s[i]]++;
136     f[0] = b[0];
137     FOR (i, 1, M) f[i] = f[i - 1] + b[i];
138     FORD (i, bs - 1, -1) sa[--f[s[p[i]]]] = p[i];
139     FOR (i, 1, M) f[i] = f[i - 1] + b[i - 1];
140     FOR (i, 0, n) if (sa[i] > 0 && !t[sa[i] - 1]) sa[f[s[sa[i] - 1]]++] = sa[i] - 1;
141     f[0] = b[0];
142     FOR (i, 1, M) f[i] = f[i - 1] + b[i];
143 }
144 FORD (i, n - 1, -1) if (sa[i] > 0 && t[sa[i] - 1]) sa[--f[s[sa[i] - 1]]] = sa[i] - 1;
145 }
146 template<class T>
147 inline void sais(T s, int *sa, int n, bool *t, int *b, int *c, int M) {
148     int i, j, bs = 0, cnt = 0, p = -1, x, *r = b + M;
149     t[n - 1] = 1;
150     FORD (i, n - 2, -1) t[i] = s[i] < s[i + 1] || (s[i] == s[i + 1] && t[i + 1]);
151     FOR (i, 1, n) if (t[i] && !t[i - 1]) c[bs++] = i;
152     inducedSort(s, sa, n, M, bs, t, b, r, c);
153     for (i = bs = 0; i < n; i++) if (isLMS(sa[i], t)) sa[bs++] = sa[i];
154     FOR (i, bs, n) sa[i] = -1;
155     FOR (i, 0, bs) {
156         x = sa[i];
157         for (j = 0; j < n; j++) {
158             if (p == -1 || s[x + j] != s[p + j] || t[x + j] != t[p + j]) { cnt++; p = x; break; }
159             else if (j > 0 && (isLMS(x + j, t) || isLMS(p + j, t))) break;
160         }
161         x = (~x & 1 ? x >> 1 : x - 1 >> 1), sa[bs + x] = cnt - 1;
162     }
163     for (i = j = n - 1; i >= bs; i--) if (sa[i] >= 0) sa[j--] = sa[i];
164     int *sl = sa + n - bs, *d = c + bs;
165     if (cnt < bs) sais(sl, sa, bs, t + n, b, c + bs, cnt);
166     else FOR (i, 0, bs) sa[sl[i]] = i;
167     FOR (i, 0, bs) d[i] = c[sa[i]];
168     inducedSort(s, sa, n, M, bs, t, b, r, d);
169 }
170 template<typename T>
171 inline void getHeight(T s, const int n, const int *sa) {
172     for (int i = 0, k = 0; i < n; i++) {
173         if (rk[i] == 0) k = 0;
174         else {
175             if (k > 0) k--;
176             int j = sa[rk[i] - 1];
177             while (i + k < n && j + k < n && s[i + k] == s[j + k]) k++;
178             ht[rk[i]] = k;
179         }
180     }
181 }
182 template<class T>
183 inline void init(T s, int n, int M) {
184     sais(s, sa, n, t, b, c, M);
185     for (int i = 1; i < n; i++) rk[sa[i]] = i;
186     getHeight(s, n, sa);
187 }

```



```

187 }
188 };
189 SuffixArray<N> sa;
190 int main() {
191     int n = s.length();
192     sa.init(s, n, 128);
193     FOR (i, 1, n + 1) printf("%d%c", sa.sa[i]
194         + 1, i == _i - 1 ? '\n' : ' ');
195     FOR (i, 2, n + 1) printf("%d%c", sa.ht[i]
196         , i == _i - 1 ? '\n' : ' ');
197 }

```

6.7 Suffix Automation

```

1 namespace sam {
2     const int M = N << 1;
3     int t[M][26], len[M] = {-1}, fa[M], sz =
4         2, last = 1;
5     void init() { memset(t, 0, (sz + 10) *
6         sizeof t[0]); sz = 2; last = 1; }
7     void ins(int ch) {
8         int p = last, np = last = sz++;
9         len[np] = len[p] + 1;
10        for (; p && !t[p][ch]; p = fa[p]) t[p]
11            [ch] = np;
12        if (!p) { fa[np] = 1; return; }
13        int q = t[p][ch];
14        if (len[p] + 1 == len[q]) fa[np] = q;
15        else {
16            int nq = sz++; len[nq] = len[p] +
17                1;
18            memcpy(t[nq], t[q], sizeof t[0]);
19            fa[nq] = fa[q];
20            fa[np] = fa[q] = nq;
21            for (; t[p][ch] == q; p = fa[p])
22                t[p][ch] = nq;
23        }
24    }
25    int c[M] = {1}, a[M];
26    void rsort() {
27        FOR (i, 1, sz) c[i] = 0;
28        FOR (i, 1, sz) c[len[i]]++;
29        FOR (i, 1, sz) c[i] += c[i - 1];
30        FOR (i, 1, sz) a[--c[len[i]]] = i;
31    }
32    // really-generalized sam
33    int t[M][26], len[M] = {-1}, fa[M], sz = 2,
34        last = 1;
35    LL cnt[M][2];
36    void ins(int ch, int id) {
37        int p = last, np = 0, nq = 0, q = -1;
38        if (!t[p][ch]) {
39            np = sz++;
40            len[np] = len[p] + 1;
41            for (; p && !t[p][ch]; p = fa[p]) t[p]
42                [ch] = np;
43        }
44        if (!p) fa[np] = 1;
45        else {
46            q = t[p][ch];
47            if (len[p] + 1 == len[q]) fa[np] = q;
48            else {
49                nq = sz++; len[nq] = len[p] + 1;
50                one[nq] = one[q];
51                memcpy(t[nq], t[q], sizeof t[0]);
52                fa[nq] = fa[q];
53                fa[np] = fa[q] = nq;
54                for (; t[p][ch] == q; p = fa[p])
55                    t[p][ch] = nq;
56            }
57        }
58        // lexicographical order
59        // generalized sam
60        int up[M], c[256] = {2}, a[M];
61        void rsort2() {
62            FOR (i, 1, 256) c[i] = 0;
63            FOR (i, 2, sz) up[i] = s[one[i] + len[fa[
64                i]]];
65            FOR (i, 2, sz) c[up[i]]++;
66            FOR (i, 1, 256) c[i] += c[i - 1];
67            FOR (i, 2, sz) a[--c[up[i]]] = i;
68            FOR (i, 2, sz) G[fa[a[i]]].push_back(a[i]
69                );
70        }
71        int t[M][26], len[M] = {0}, fa[M], sz = 2,
72            last = 1;
73        char* one[M];
74        void ins(int ch, char* pp) {
75            int p = last, np = 0, nq = 0, q = -1;
76            if (!t[p][ch]) {
77                np = sz++; one[np] = pp;
78                len[np] = len[p] + 1;
79                for (; p && !t[p][ch]; p = fa[p]) t[p]
80                    [ch] = np;
81            }
82            if (!p) fa[np] = 1;
83            else {
84                q = t[p][ch];
85                if (len[p] + 1 == len[q]) fa[np] = q;
86                else {
87                    nq = sz++; len[nq] = len[p] + 1;
88                    one[nq] = one[q];
89                    memcpy(t[nq], t[q], sizeof t[0]);
90                    fa[nq] = fa[q];
91                    fa[np] = fa[q] = nq;
92                }
93            }
94        }
95    }
96    memcpy(t[nq], t[q], sizeof t[0]);
97    fa[nq] = fa[q];
98    fa[np] = fa[q] = nq;
99    for (; t[p][ch] == q; p = fa[p])
100        t[p][ch] = nq;
101    }
102    last = np ? np : nq ? nq : q;
103    cnt[last][id] = 1;
104    // lexicographical order
105    // rsort2 is not topo sort
106    void ins(int ch, int pp) {
107        int p = last, np = last = sz++;
108        len[np] = len[p] + 1; one[np] = pos[np] =
109            pp;
110        for (; p && !t[p][ch]; p = fa[p]) t[p][ch]
111            = np;
112        if (!p) { fa[np] = 1; return; }
113        int q = t[p][ch];
114        if (len[q] == len[p] + 1) fa[np] = q;
115        else {
116            int nq = sz++; len[nq] = len[p] + 1;
117            one[nq] = one[q];
118            memcpy(t[nq], t[q], sizeof t[0]);
119            fa[nq] = fa[q];
120            fa[q] = fa[np] = nq;
121            for (; p && t[p][ch] == q; p = fa[p])
122                t[p][ch] = nq;
123        }
124        // lexicographical order
125        // generalized sam
126        int up[M], c[256] = {2}, a[M];
127        void rsort2() {
128            FOR (i, 1, 256) c[i] = 0;
129            FOR (i, 2, sz) up[i] = s[one[i] + len[fa[
130                i]]];
131            FOR (i, 2, sz) c[up[i]]++;
132            FOR (i, 1, 256) c[i] += c[i - 1];
133            FOR (i, 2, sz) a[--c[up[i]]] = i;
134            FOR (i, 2, sz) G[fa[a[i]]].push_back(a[i]
135                );
136        }
137        int t[M][26], len[M] = {0}, fa[M], sz = 2,
138            last = 1;
139        char* one[M];
140        void ins(int ch, char* pp) {
141            int p = last, np = 0, nq = 0, q = -1;
142            if (!t[p][ch]) {
143                np = sz++; one[np] = pp;
144                len[np] = len[p] + 1;
145                for (; p && !t[p][ch]; p = fa[p]) t[p]
146                    [ch] = np;
147            }
148            if (!p) fa[np] = 1;
149            else {
150                q = t[p][ch];
151                if (len[p] + 1 == len[q]) fa[np] = q;
152                else {
153                    nq = sz++; len[nq] = len[p] + 1;
154                    one[nq] = one[q];
155                    memcpy(t[nq], t[q], sizeof t[0]);
156                    fa[nq] = fa[q];
157                    fa[np] = fa[q] = nq;
158                }
159            }
160        }
161        memcpy(t[nq], t[q], sizeof t[0]);
162        fa[nq] = fa[q];
163        fa[np] = fa[q] = nq;
164        for (; t[p][ch] == q; p = fa[p])
165            t[p][ch] = nq;
166        last = np ? np : nq ? nq : q;
167        cnt[last][id] = 1;
168        // lexicographical order
169        // rsort2 is not topo sort
170        void ins(int ch, int pp) {
171            int p = last, np = last = sz++;
172            len[np] = len[p] + 1; one[np] = pos[np] =
173                pp;
174            for (; p && !t[p][ch]; p = fa[p]) t[p][ch]
175                = np;
176            if (!p) { fa[np] = 1; return; }
177            int q = t[p][ch];
178            if (len[q] == len[p] + 1) fa[np] = q;
179            else {
180                int nq = sz++; len[nq] = len[p] + 1;
181                one[nq] = one[q];
182                memcpy(t[nq], t[q], sizeof t[0]);
183                fa[nq] = fa[q];
184                fa[q] = fa[np] = nq;
185                for (; p && t[p][ch] == q; p = fa[p])
186                    t[p][ch] = nq;
187            }
188        }
189        // lexicographical order
190        // generalized sam
191        int up[M], c[256] = {2}, a[M];
192        void rsort2() {
193            FOR (i, 1, 256) c[i] = 0;
194            FOR (i, 2, sz) up[i] = s[one[i] + len[fa[
195                i]]];
196            FOR (i, 2, sz) c[up[i]]++;
197            FOR (i, 1, 256) c[i] += c[i - 1];
198            FOR (i, 2, sz) a[--c[up[i]]] = i;
199            FOR (i, 2, sz) G[fa[a[i]]].push_back(a[i]
200                );
201        }
202        int t[M][26], len[M] = {0}, fa[M], sz = 2,
203            last = 1;
204        char* one[M];
205        void ins(int ch, char* pp) {
206            int p = last, np = 0, nq = 0, q = -1;
207            if (!t[p][ch]) {
208                np = sz++; one[np] = pp;
209                len[np] = len[p] + 1;
210                for (; p && !t[p][ch]; p = fa[p]) t[p]
211                    [ch] = np;
212            }
213            if (!p) fa[np] = 1;
214            else {
215                q = t[p][ch];
216                if (len[p] + 1 == len[q]) fa[np] = q;
217                else {
218                    nq = sz++; len[nq] = len[p] + 1;
219                    one[nq] = one[q];
220                    memcpy(t[nq], t[q], sizeof t[0]);
221                    fa[nq] = fa[q];
222                    fa[q] = fa[np] = nq;
223                    for (; p && t[p][ch] == q; p = fa[p])
224                        t[p][ch] = nq;
225                }
226            }
227        }
228        memcpy(t[nq], t[q], sizeof t[0]);
229        fa[nq] = fa[q];
230        fa[np] = fa[q] = nq;
231        for (; t[p][ch] == q; p = fa[p])
232            t[p][ch] = nq;
233        last = np ? np : nq ? nq : q;
234        cnt[last][id] = 1;
235        // lexicographical order
236        // rsort2 is not topo sort
237        void ins(int ch, int pp) {
238            int p = last, np = last = sz++;
239            len[np] = len[p] + 1; one[np] = pos[np] =
240                pp;
241            for (; p && !t[p][ch]; p = fa[p]) t[p][ch]
242                = np;
243            if (!p) { fa[np] = 1; return; }
244            int q = t[p][ch];
245            if (len[q] == len[p] + 1) fa[np] = q;
246            else {
247                int nq = sz++; len[nq] = len[p] + 1;
248                one[nq] = one[q];
249                memcpy(t[nq], t[q], sizeof t[0]);
250                fa[nq] = fa[q];
251                fa[q] = fa[np] = nq;
252                for (; p && t[p][ch] == q; p = fa[p])
253                    t[p][ch] = nq;
254            }
255        }
256        // lexicographical order
257        // generalized sam
258        int up[M], c[256] = {2}, a[M];
259        void rsort2() {
260            FOR (i, 1, 256) c[i] = 0;
261            FOR (i, 2, sz) up[i] = s[one[i] + len[fa[
262                i]]];
263            FOR (i, 2, sz) c[up[i]]++;
264            FOR (i, 1, 256) c[i] += c[i - 1];
265            FOR (i, 2, sz) a[--c[up[i]]] = i;
266            FOR (i, 2, sz) G[fa[a[i]]].push_back(a[i]
267                );
268        }
269        int t[M][26], len[M] = {0}, fa[M], sz = 2,
270            last = 1;
271        char* one[M];
272        void ins(int ch, char* pp) {
273            int p = last, np = 0, nq = 0, q = -1;
274            if (!t[p][ch]) {
275                np = sz++; one[np] = pp;
276                len[np] = len[p] + 1;
277                for (; p && !t[p][ch]; p = fa[p]) t[p]
278                    [ch] = np;
279            }
280            if (!p) fa[np] = 1;
281            else {
282                q = t[p][ch];
283                if (len[p] + 1 == len[q]) fa[np] = q;
284                else {
285                    nq = sz++; len[nq] = len[p] + 1;
286                    one[nq] = one[q];
287                    memcpy(t[nq], t[q], sizeof t[0]);
288                    fa[nq] = fa[q];
289                    fa[q] = fa[np] = nq;
290                    for (; p && t[p][ch] == q; p = fa[p])
291                        t[p][ch] = nq;
292                }
293            }
294        }
295        memcpy(t[nq], t[q], sizeof t[0]);
296        fa[nq] = fa[q];
297        fa[np] = fa[q] = nq;
298        for (; t[p][ch] == q; p = fa[p])
299            t[p][ch] = nq;
300        last = np ? np : nq ? nq : q;
301        cnt[last][id] = 1;
302        // lexicographical order
303        // rsort2 is not topo sort
304        void ins(int ch, int pp) {
305            int p = last, np = last = sz++;
306            len[np] = len[p] + 1; one[np] = pos[np] =
307                pp;
308            for (; p && !t[p][ch]; p = fa[p]) t[p][ch]
309                = np;
310            if (!p) { fa[np] = 1; return; }
311            int q = t[p][ch];
312            if (len[q] == len[p] + 1) fa[np] = q;
313            else {
314                int nq = sz++; len[nq] = len[p] + 1;
315                one[nq] = one[q];
316                memcpy(t[nq], t[q], sizeof t[0]);
317                fa[nq] = fa[q];
318                fa[q] = fa[np] = nq;
319                for (; p && t[p][ch] == q; p = fa[p])
320                    t[p][ch] = nq;
321            }
322        }
323        // lexicographical order
324        // generalized sam
325        int up[M], c[256] = {2}, a[M];
326        void rsort2() {
327            FOR (i, 1, 256) c[i] = 0;
328            FOR (i, 2, sz) up[i] = s[one[i] + len[fa[
329                i]]];
330            FOR (i, 2, sz) c[up[i]]++;
331            FOR (i, 1, 256) c[i] += c[i - 1];
332            FOR (i, 2, sz) a[--c[up[i]]] = i;
333            FOR (i, 2, sz) G[fa[a[i]]].push_back(a[i]
334                );
335        }
336        int t[M][26], len[M] = {0}, fa[M], sz = 2,
337            last = 1;
338        char* one[M];
339        void ins(int ch, char* pp) {
340            int p = last, np = 0, nq = 0, q = -1;
341            if (!t[p][ch]) {
342                np = sz++; one[np] = pp;
343                len[np] = len[p] + 1;
344                for (; p && !t[p][ch]; p = fa[p]) t[p]
345                    [ch] = np;
346            }
347            if (!p) fa[np] = 1;
348            else {
349                q = t[p][ch];
350                if (len[p] + 1 == len[q]) fa[np] = q;
351                else {
352                    nq = sz++; len[nq] = len[p] + 1;
353                    one[nq] = one[q];
354                    memcpy(t[nq], t[q], sizeof t[0]);
355                    fa[nq] = fa[q];
356                    fa[q] = fa[np] = nq;
357                    for (; p && t[p][ch] == q; p = fa[p])
358                        t[p][ch] = nq;
359                }
360            }
361        }
362        memcpy(t[nq], t[q], sizeof t[0]);
363        fa[nq] = fa[q];
364        fa[np] = fa[q] = nq;
365        for (; t[p][ch] == q; p = fa[p])
366            t[p][ch] = nq;
367        last = np ? np : nq ? nq : q;
368        cnt[last][id] = 1;
369        // lexicographical order
370        // rsort2 is not topo sort
371        void ins(int ch, int pp) {
372            int p = last, np = last = sz++;
373            len[np] = len[p] + 1; one[np] = pos[np] =
374                pp;
375            for (; p && !t[p][ch]; p = fa[p]) t[p][ch]
376                = np;
377            if (!p) { fa[np] = 1; return; }
378            int q = t[p][ch];
379            if (len[q] == len[p] + 1) fa[np] = q;
380            else {
381                int nq = sz++; len[nq] = len[p] + 1;
382                one[nq] = one[q];
383                memcpy(t[nq], t[q], sizeof t[0]);
384                fa[nq] = fa[q];
385                fa[q] = fa[np] = nq;
386                for (; p && t[p][ch] == q; p = fa[p])
387                    t[p][ch] = nq;
388            }
389        }
390        // lexicographical order
391        // generalized sam
392        int up[M], c[256] = {2}, a[M];
393        void rsort2() {
394            FOR (i, 1, 256) c[i] = 0;
395            FOR (i, 2, sz) up[i] = s[one[i] + len[fa[
396                i]]];
397            FOR (i, 2, sz) c[up[i]]++;
398            FOR (i, 1, 256) c[i] += c[i - 1];
399            FOR (i, 2, sz) a[--c[up[i]]] = i;
400            FOR (i, 2, sz) G[fa[a[i]]].push_back(a[i]
401                );
402        }
403        int t[M][26], len[M] = {0}, fa[M], sz = 2,
404            last = 1;
405        char* one[M];
406        void ins(int ch, char* pp) {
407            int p = last, np = 0, nq = 0, q = -1;
408            if (!t[p][ch]) {
409                np = sz++; one[np] = pp;
410                len[np] = len[p] + 1;
411                for (; p && !t[p][ch]; p = fa[p]) t[p]
412                    [ch] = np;
413            }
414            if (!p) fa[np] = 1;
415            else {
416                q = t[p][ch];
417                if (len[p] + 1 == len[q]) fa[np] = q;
418                else {
419                    nq = sz++; len[nq] = len[p] + 1;
420                    one[nq] = one[q];
421                    memcpy(t[nq], t[q], sizeof t[0]);
422                    fa[nq] = fa[q];
423                    fa[q] = fa[np] = nq;
424                    for (; p && t[p][ch] == q; p = fa[p])
425                        t[p][ch] = nq;
426                }
427            }
428        }
429        memcpy(t[nq], t[q], sizeof t[0]);
430        fa[nq] = fa[q];
431        fa[np] = fa[q] = nq;
432        for (; t[p][ch] == q; p = fa[p])
433            t[p][ch] = nq;
434        last = np ? np : nq ? nq : q;
435        cnt[last][id] = 1;
436        // lexicographical order
437        // rsort2 is not topo sort
438        void ins(int ch, int pp) {
439            int p = last, np = last = sz++;
440            len[np] = len[p] + 1; one[np] = pos[np] =
441                pp;
442            for (; p && !t[p][ch]; p = fa[p]) t[p][ch]
443                = np;
444            if (!p) { fa[np] = 1; return; }
445            int q = t[p][ch];
446            if (len[q] == len[p] + 1) fa[np] = q;
447            else {
448                int nq = sz++; len[nq] = len[p] + 1;
449                one[nq] = one[q];
450                memcpy(t[nq], t[q], sizeof t[0]);
451                fa[nq] = fa[q];
452                fa[q] = fa[np] = nq;
453                for (; p && t[p][ch] == q; p = fa[p])
454                    t[p][ch] = nq;
455            }
456        }
457        // lexicographical order
458        // generalized sam
459        int up[M], c[256] = {2}, a[M];
460        void rsort2() {
461            FOR (i, 1, 256) c[i] = 0;
462            FOR (i, 2, sz) up[i] = s[one[i] + len[fa[
463                i]]];
464            FOR (i, 2, sz) c[up[i]]++;
465            FOR (i, 1, 256) c[i] += c[i - 1];
466            FOR (i, 2, sz) a[--c[up[i]]] = i;
467            FOR (i, 2, sz) G[fa[a[i]]].push_back(a[i]
468                );
469        }
470        int t[M][26], len[M] = {0}, fa[M], sz = 2,
471            last = 1;
472        char* one[M];
473        void ins(int ch, char* pp) {
474            int p = last, np = 0, nq = 0, q = -1;
475            if (!t[p][ch]) {
476                np = sz++; one[np] = pp;
477                len[np] = len[p] + 1;
478                for (; p && !t[p][ch]; p = fa[p]) t[p]
479                    [ch] = np;
480            }
481            if (!p) fa[np] = 1;
482            else {
483                q = t[p][ch];
484                if (len[p] + 1 == len[q]) fa[np] = q;
485                else {
486                    nq = sz++; len[nq] = len[p] + 1;
487                    one[nq] = one[q];
488                    memcpy(t[nq], t[q], sizeof t[0]);
489                    fa[nq] = fa[q];
490                    fa[q] = fa[np] = nq;
491                    for (; p && t[p][ch] == q; p = fa[p])
492                        t[p][ch] = nq;
493                }
494            }
495        }
496        memcpy(t[nq], t[q], sizeof t[0]);
497        fa[nq] = fa[q];
498        fa[np] = fa[q] = nq;
499        for (; t[p][ch] == q; p = fa[p])
500            t[p][ch] = nq;
501        last = np ? np : nq ? nq : q;
502        cnt[last][id] = 1;
503        // lexicographical order
504        // rsort2 is not topo sort
505        void ins(int ch, int pp) {
506            int p = last, np = last = sz++;
507            len[np] = len[p] + 1; one[np] = pos[np] =
508                pp;
509            for (; p && !t[p][ch]; p = fa[p]) t[p][ch]
510                = np;
511            if (!p) { fa[np] = 1; return; }
512            int q = t[p][ch];
513            if (len[q] == len[p] + 1) fa[np] = q;
514            else {
515                int nq = sz++; len[nq] = len[p] + 1;
516                one[nq] = one[q];
517                memcpy(t[nq], t[q], sizeof t[0]);
518                fa[nq] = fa[q];
519                fa[q] = fa[np] = nq;
520                for (; p && t[p][ch] == q; p = fa[p])
521                    t[p][ch] = nq;
522            }
523        }
524        // lexicographical order
525        // generalized sam
526        int up[M], c[256] = {2}, a[M];
527        void rsort2() {
528            FOR (i, 1, 256) c[i] = 0;
529            FOR (i, 2, sz) up[i] = s[one[i] + len[fa[
530                i]]];
531            FOR (i, 2, sz) c[up[i]]++;
532            FOR (i, 1, 256) c[i] += c[i - 1];
533            FOR (i, 2, sz) a[--c[up[i]]] = i;
534            FOR (i, 2, sz) G[fa[a[i]]].push_back(a[i]
535                );
536        }
537        int t[M][26], len[M] = {0}, fa[M], sz = 2,
538            last = 1;
539        char* one[M];
540        void ins(int ch, char* pp) {
541            int p = last, np = 0, nq = 0, q = -1;
542            if (!t[p][ch]) {
543                np = sz++; one[np] = pp;
544                len[np] = len[p] + 1;
545                for (; p && !t[p][ch]; p = fa[p]) t[p]
546                    [ch] = np;
547            }
548            if (!p) fa[np] = 1;
549            else {
550                q = t[p][ch];
551                if (len[p] + 1 == len[q]) fa[np] = q;
552                else {
553                    nq = sz++; len[nq] = len[p] + 1;
554                    one[nq] = one[q];
555                    memcpy(t[nq], t[q], sizeof t[0]);
556                    fa[nq] = fa[q];
557                    fa[q] = fa[np] = nq;
558                    for (; p && t[p][ch] == q; p = fa[p])
559                        t[p][ch] = nq;
560                }
561            }
562        }
563        memcpy(t[nq], t[q], sizeof t[0]);
564        fa[nq] = fa[q];
565        fa[np] = fa[q] = nq;
566        for (; t[p][ch] == q; p = fa[p])
567            t[p][ch] = nq;
568        last = np ? np : nq ? nq : q;
569        cnt[last][id] = 1;
570        // lexicographical order
571        // rsort2 is not topo sort
572        void ins(int ch, int pp) {
573            int p = last, np = last = sz++;
574            len[np] = len[p] + 1; one[np] = pos[np] =
575                pp;
576            for (; p && !t[p][ch]; p = fa[p]) t[p][ch]
577                = np;
578            if (!p) { fa[np] = 1; return; }
579            int q = t[p][ch];
580            if (len[q] == len[p] + 1) fa[np] = q;
581            else {
582                int nq = sz++; len[nq] = len[p] + 1;
583                one[nq] = one[q];
584                memcpy(t[nq], t[q], sizeof t[0]);
585                fa[nq] = fa[q];
586                fa[q] = fa[np] = nq;
587                for (; p && t[p][ch] == q; p = fa[p])
588                    t[p][ch] = nq;
589            }
590        }
591        // lexicographical order
592        // generalized sam
593        int up[M], c[256] = {2}, a[M];
594        void rsort2() {
595            FOR (i, 1, 256) c[i] = 0;
596            FOR (i, 2, sz) up[i] = s[one[i] + len[fa[
597                i]]];
598            FOR (i, 2, sz) c[up[i]]++;
599            FOR (i, 1, 256) c[i] += c[i - 1];
600            FOR (i, 2, sz) a[--c[up[i]]] = i;
601            FOR (i, 2, sz) G[fa[a[i]]].push_back(a[i]
602                );
603        }
604        int t[M][26], len[M] = {0}, fa[M], sz = 2,
605            last = 1;
606        char* one[M];
607        void ins(int ch, char* pp) {
608            int p = last, np = 0, nq = 0, q = -1;
609            if (!t[p][ch]) {
610                np = sz++; one[np] = pp;
611                len[np] = len[p] + 1;
612                for (; p && !t[p][ch]; p = fa[p]) t[p]
613                    [ch] = np;
614            }
615            if (!p) fa[np] = 1;
616            else {
617                q = t[p][ch];
618                if (len[p] + 1 == len[q]) fa[np] = q;
619                else {
620                    nq = sz++; len[nq] = len[p] + 1;
621                    one[nq] = one[q];
622                    memcpy(t[nq], t[q], sizeof t[0]);
623                    fa[nq] = fa[q];
624                    fa[q] = fa[np] = nq;
625                    for (; p && t[p][ch] == q; p = fa[p])
626                        t[p][ch] = nq;
627                }
628            }
629        }
630        memcpy(t[nq], t[q], sizeof t[0]);
631        fa[nq] = fa[q];
632        fa[np] = fa[q] = nq;
633        for (; t[p][ch] == q; p = fa[p])
634            t[p][ch] = nq;
635        last = np ? np : nq ? nq : q;
636        cnt[last][id] = 1;
637        // lexicographical order
638        // rsort2 is not topo sort
639        void ins(int ch, int pp) {
640            int p = last, np = last = sz++;
641            len[np] = len[p] + 1; one[np] = pos[np] =
642                pp;
643            for (; p && !t[p][ch]; p = fa[p]) t[p][ch]
644                = np;
645            if (!p) { fa[np] = 1; return; }
646            int q = t[p][ch];
647            if (len[q] == len[p] + 1) fa[np] = q;
648            else {
649                int nq = sz++; len[nq] = len[p] + 1;
650                one[nq] = one[q];
651                memcpy(t[nq], t[q], sizeof t[0]);
652                fa[nq] = fa[q];
653                fa[q] = fa[np] = nq;
654                for (; p && t[p][ch] == q; p = fa[p])
655                    t[p][ch] = nq;
656            }
657        }
658        // lexicographical order
659        // generalized sam
660        int up[M], c[256] = {2}, a[M];
661        void rsort2() {
662            FOR (i, 1, 256) c[i] = 0;
663            FOR (i, 2, sz) up[i] = s[one[i] + len[fa[
664                i]]];
665            FOR (i, 2, sz) c[up[i]]++;
666            FOR (i, 1, 256) c[i] += c[i - 1];
667            FOR (i, 2, sz) a[--c[up[i]]] = i;
668            FOR (i, 2, sz) G[fa[a[i]]].push_back(a[i]
669                );
670        }
671        int t[M][26], len[M] = {0}, fa[M], sz = 2,
672            last = 1;
673        char* one[M];
674        void ins(int ch, char* pp) {
675            int p = last, np = 0, nq = 0, q = -1;
676            if (!t[p][ch]) {
677                np = sz++; one[np] = pp;
678                len[np] = len[p] + 1;
679                for (; p && !t[p][ch]; p = fa[p]) t[p]
680                    [ch] = np;
681            }
682            if (!p) fa[np] = 1;
683            else {
684                q = t[p][ch];
685                if (len[p] + 1 == len[q]) fa[np] = q;
686                else {
687                    nq = sz++; len[nq] = len[p] + 1;
688                    one[nq] = one[q];
689                    memcpy(t[nq], t[q], sizeof t[0]);
690                    fa[nq] = fa[q];
691                    fa[q] = fa[np] = nq;
692                    for (; p && t[p][ch] == q; p = fa[p])
693                        t[p][ch] = nq;
694                }
695            }
696        }
697        memcpy(t[nq], t[q], sizeof t[0]);
698        fa[nq] = fa[q];
699        fa[np] = fa[q] = nq;
700        for (; t[p][ch] == q; p = fa[p])
701            t[p][ch] = nq;
702        last = np ? np : nq ? nq : q;
703        cnt[last][id] = 1;
704        // lexicographical order
705        // rsort2 is not topo sort
706        void ins(int ch, int pp) {
707            int p = last, np = last = sz++;
708            len[np] = len[p] + 1; one[np] = pos[np] =
709                pp;
710            for (; p && !t[p][ch]; p = fa[p]) t[p][ch]
711                = np;
712            if (!p) { fa[np] = 1; return; }
713            int q = t[p][ch];
714            if (len[q] == len[p] + 1) fa[np] = q;
715            else {
716                int nq = sz++; len[nq] = len[p] + 1;
717                one[nq] = one[q];
718                memcpy(t[nq], t[q], sizeof t[0]);
719                fa[nq] = fa[q];
720                fa[q] = fa[np] = nq;
721                for (; p && t[p][ch] == q; p = fa[p])
722                    t[p][ch] = nq;
723            }
724        }
725        // lexicographical order
726        // generalized sam
727        int up[M], c[256] = {2}, a[M];
728        void rsort2() {
729            FOR (i, 1, 256) c[i] = 0;
730            FOR (i, 2, sz) up[i] = s[one[i] + len[fa[
731                i]]];
732            FOR (i, 2, sz) c[up[i]]++;
733            FOR (i, 1, 256) c[i] += c[i - 1];
734            FOR (i, 2, sz) a[--c[up[i]]] = i;
735            FOR (i, 2, sz) G[fa[a[i]]].push_back(a[i]
736                );
737        }
738        int t[M][26], len[M] = {0}, fa[M], sz = 2,
739            last = 1;
740        char* one[M];
741        void ins(int ch, char* pp) {
742            int p = last, np = 0, nq = 0, q = -1;
743            if (!t[p][ch]) {
744                np = sz++; one[np] = pp;
745                len[np] = len[p] + 1;
746                for (; p && !t[p][ch]; p = fa[p]) t[p]
747                    [ch] = np;
748            }
749            if (!p) fa[np] = 1;
750            else {
751                q = t[p][ch];
752                if (len[p] + 1 == len[q]) fa[np] = q;
753                else {
754                    nq = sz++; len[nq] = len[p] + 1;
755                    one[nq] = one[q];
756                    memcpy(t[nq], t[q], sizeof t[0]);
757                    fa[nq] = fa[q];
758                    fa[q] = fa[np] = nq;
759                    for (; p && t[p][ch] == q; p = fa[p])
760                        t[p][ch] = nq;
761                }
762            }
763        }
764        memcpy(t[nq], t[q], sizeof t[0]);
765        fa[nq] = fa[q];
766        fa[np] = fa[q] = nq;
767        for (; t[p][ch] == q; p = fa[p])
768            t[p][ch] = nq;
769        last = np ? np : nq ? nq : q;
770        cnt[last][id] = 1;
771        // lexicographical order
772        // rsort2 is not topo sort
773        void ins(int ch, int pp) {
774            int p = last, np = last = sz++;
775            len[np] = len[p] + 1; one[np] = pos[np] =
776                pp;
777            for (; p && !t[p][ch]; p = fa[p]) t[p][ch]
778                = np;
779            if (!p) { fa[np] = 1; return; }
780            int q = t[p][ch];
781            if (len[q] == len[p] + 1) fa[np] = q;
782            else {
783                int nq = sz++; len[nq] = len[p] + 1;
784                one[nq] = one[q];
785                memcpy(t[nq], t[q], sizeof t[0]);
786                fa[nq] = fa[q];
787                fa[q] = fa[np] = nq;
788                for (; p && t[p][ch] == q; p = fa[p])
789                    t[p][ch] = nq;
790            }
791        }
7
```

```

156     if (t != null) t->fa = f; f->c(dd) =
157         t;
158     o->c(!dd) = f->up(); f->fa = o;
159 }
160 void splay(P* o) {
161     o->all_down();
162     while (o->has_fa()) {
163         if (o->fa->has_fa()) {
164             rot(o->d() ^ o->fa->d() ? o :
165                 o->fa);
166             rot(o);
167         }
168         o->up();
169     }
170 void access(int last, P* u, P* v = null)
171 {
172     if (u == null) { v->last = last;
173         return; }
174     splay(u);
175     P *t = u;
176     while (t->ls != null) t = t->ls;
177     int L = len[fa[t - pool]] + 1, R =
178         len[u - pool];
179
180     if (u->last) bit::add(u->last - R +
181         2, u->last - L + 2, 1);
182     else bit::add(1, 1, R - L + 1);
183     bit::add(last - R + 2, last - L + 2,
184         -1);
185
186     u->rs = v;
187     access(last, u->up()->fa, u);
188 }
189 void insert(P* u, P* v, P* t) {
190     if (v != null) { splay(v); v->rs =
191         null; }
192     splay(u);
193     u->fa = t; t->fa = v;
194 }
195 void ins(int ch, int pp) {
196     int p = last, np = last = sz++;
197     len[np] = len[p] + 1;
198     for (; p && !t[p][ch]; p = fa[p]) t[p]
199         [ch] = np;
200     if (!p) fa[np] = 1;
201     else {
202         int q = t[p][ch];
203         if (len[p] + 1 == len[q]) { fa[np]
204             = q; G[np]->fa = G[q]; }
205         else {
206             int nq = sz++; len[nq] = len[
207                 p] + 1;
208             memcpy(t[nq], t[q], sizeof t
209                 [0]);
210             insert(G[q], G[fa[q]], G[nq])
211                 ;
212             G[nq]->last = G[q]->last;
213             fa[nq] = fa[q];
214             fa[np] = fa[q] = nq;
215             G[np]->fa = G[nq];
216             for (; t[p][ch] == q; p = fa[
217                 p]) t[p][ch] = nq;
218         }
219     }
220     access(pp + 1, G[np]);

```

```

208     }
209 void init() {
210     ++pit;
211     FOR (i, 1, N) {
212         G[i] = pit++;
213         G[i]->ls = G[i]->rs = G[i]->fa =
214             null;
215     }
216     G[1] = null;
217 }
218 }

```

7 Miscellaneous

7.1 Date

```

1 // Routines for performing computations on
2 // dates. In these
3 // routines, months are expressed as integers
4 // from 1 to 12, days
5 // are expressed as integers from 1 to 31,
6 // and
7 // years are expressed as 4-digit integers.
8 string dayOfWeek[] = {"Mo", "Tu", "We", "Th",
9     "Fr", "Sa", "Su"};
10 // converts Gregorian date to integer (Julian
11 // day number)
12 int DateToInt (int m, int d, int y){
13     return
14         1461 * (y + 4800 + (m - 14) / 12) / 4 +
15         367 * (m - 2 - (m - 14) / 12 * 12) / 12 -
16         3 * ((y + 4900 + (m - 14) / 12) / 100) /
17         4 +
18         d - 32075;
19 }
20 // converts integer (Julian day number) to
21 // Gregorian date: month/day/year
22 void IntToDate (int jd, int &m, int &d, int &y){
23     int x, n, i, j;
24     x = jd + 68569;
25     n = 4 * x / 146097;
26     x -= (146097 * n + 3) / 4;
27     i = (4000 * (x + 1)) / 1461001;
28     x -= 1461 * i / 4 - 31;
29     j = 80 * x / 2447;
30     d = x - 2447 * j / 80;
31     x = j / 11;
32     m = j + 2 - 12 * x;
33     y = 100 * (n - 49) + i + x;
34 }
35 // converts integer (Julian day number) to
36 // day of week
37 string IntToDay (int jd){
38     return dayOfWeek[jd % 7];
39 }

```

7.2 Subset Enumeration

```

1 // all proper subset
2 for (int s = (S - 1) & S; s; s = (s - 1) & S)
3     {
4         // ...
5     }
6 // subset of length k
7 template<typename T>
8 void subset(int k, int n, T&& f) {
9     int t = (1 << k) - 1;
10    while (t < 1 << n) {
11        f(t);
12        int x = t & -t, y = t + x;
13        t = ((t & ~y) / x >> 1) | y;
14    }
15 }

```

7.3 Digit DP

```

1 LL dfs(LL base, LL pos, LL len, LL s, bool
2     limit) {
3     if (pos == -1) return s ? base : 1;
4     if (!limit && dp[base][pos][len][s] !=
5         -1) return dp[base][pos][len][s];
6     LL ret = 0;
7     LL ed = limit ? a[pos] : base - 1;
8     FOR (i, 0, ed + 1) {
9         tmp[pos] = i;
10        if (len == pos)
11            ret += dfs(base, pos - 1, len - (
12                i == 0), s, limit && i == a[
13                    pos]);
14        else if (s && pos < (len + 1) / 2)
15            ret += dfs(base, pos - 1, len,
16                tmp[len - pos] = i, limit &&
17                    i == a[pos]);
18        else
19            ret += dfs(base, pos - 1, len, s,
20                limit && i == a[pos]);
21    }
22    if (!limit) dp[base][pos][len][s] = ret;
23    return ret;
24 }
25 LL solve(LL x, LL base) {
26     LL sz = 0;
27     while (x) {
28         a[sz++] = x % base;
29         x /= base;
30     }
31     return dfs(base, sz - 1, sz - 1, 1, true)
32         ;
33 }

```

7.4 Simulated Annealing

```

1 // Minimum Circle Cover
2 using LD = double;
3 const int N = 1E4 + 100;
4 int x[N], y[N], n;

```

```

5 LD eval(LD xx, LD yy) {
6     LD r = 0;
7     FOR (i, 0, n)
8         r = max(r, sqrt(pow(xx - x[i], 2) +
9             pow(yy - y[i], 2)));
10    return r;
11 }
12 mt19937 mt(time(0));
13 auto rd = bind(uniform_real_distribution<LD
14 >(-1, 1), mt);
15 int main() {
16     int X, Y;
17     while (cin >> X >> Y >> n) {
18         FOR (i, 0, n) scanf("%d%d", &x[i], &y
19             [i]);
20         pair<LD, LD> ans;
21         LD M = 1e9;
22         FOR (_, 0, 100) {

```

```

LD cur_x = X / 2.0, cur_y = Y /
2.0, T = max(X, Y);
while (T > 1e-3) {
    LD best_ans = eval(cur_x,
        cur_y);
    LD best_x = cur_x, best_y =
        cur_y;
    FOR (_, 0, 20) {
        LD nxt_x = cur_x + rd() *
            T, nxt_y = cur_y +
            rd() * T;
        LD nxt_ans = eval(nxt_x,
            nxt_y);
        if (nxt_ans < best_ans) {
            best_x = nxt_x;
            best_y = nxt_y;
            best_ans = nxt_ans;
        }
    }
}

```

```

31     }
32     cur_x = best_x; cur_y =
33     best_y;
34     T *= .9;
35     if (eval(cur_x, cur_y) < M) {
36         ans = {cur_x, cur_y}; M =
37         eval(cur_x, cur_y);
38     }
39     printf("(%1f,%1f).\n%.1f\n", ans.
40         first, ans.second, eval(ans.first
41         , ans.second));

```

1 数学

1.1 杜教筛

求 $S(n) = \sum_{i=1}^n f(i)$, 其中 f 是一个积性函数。

构造一个积性函数 g , 那么由 $(f * g)(n) = \sum_{d|n} f(d)g(\frac{n}{d})$, 得到 $f(n) = (f * g)(n) - \sum_{d|n, d < n} f(d)g(\frac{n}{d})$ 。

$$\begin{aligned} g(1)S(n) &= \sum_{i=1}^n (f * g)(i) - \sum_{i=1}^n \sum_{d|i, d < i} f(d)g(\frac{n}{d}) \quad (1) \\ &\stackrel{t=\frac{i}{d}}{=} \sum_{i=1}^n (f * g)(i) - \sum_{t=2}^n g(t)S(\lfloor \frac{n}{t} \rfloor) \quad (2) \end{aligned}$$

当然, 要能够由此计算 $S(n)$, 会对 f, g 提出一些要求:

- $f * g$ 要能够快速求前缀和。
 - g 要能够快速求分段和 (前缀和)。
 - 对于正常的积性函数 $g(1) = 1$, 所以不会有什么问题。
- 在预处理 $S(n)$ 前 $n^{\frac{2}{3}}$ 项的情况下复杂度是 $O(n^{\frac{2}{3}})$ 。

1.2 素性测试

- 前置: 快速乘、快速幂
- int 范围内只需检查 2, 7, 61
- long long 范围 2, 325, 9375, 28178, 450775, 9780504, 1795265022
- 3E15 内 2, 2570940, 880937, 610386380, 4130785767
- 4E13 内 2, 2570940, 211991001, 3749873356
- <http://miller-rabin.appspot.com/>

1.3 扩展欧几里得

- 求 $ax + by = \gcd(a, b)$ 的一组解
- 如果 a 和 b 互素, 那么 x 是 a 在模 b 下的逆元
- 注意 x 和 y 可能是负数

1.4 类欧几里得

- $m = \lfloor \frac{am+b}{c} \rfloor$.
- $f(a, b, c, n) = \sum_{i=0}^n \lfloor \frac{ai+b}{c} \rfloor$: 当 $a \geq c$ or $b \geq c$ 时, $f(a, b, c, n) = (\frac{a}{c})n(n+1)/2 + (\frac{b}{c})(n+1) + f(a \bmod c, b \bmod c, c, n)$; 否则 $f(a, b, c, n) = mn - f(c, c-b-1, a, m-1)$ 。
- $g(a, b, c, n) = \sum_{i=0}^n i \lfloor \frac{ai+b}{c} \rfloor$: 当 $a \geq c$ or $b \geq c$ 时, $g(a, b, c, n) = (\frac{a}{c})n(n+1)(2n+1)/6 + (\frac{b}{c})n(n+1)/2 + g(a \bmod c, b \bmod c, c, n)$; 否则 $g(a, b, c, n) = \frac{1}{2}(n(n+1)m - f(c, c-b-1, a, m-1) - h(c, c-b-1, a, m-1))$ 。
- $h(a, b, c, n) = \sum_{i=0}^n \lfloor \frac{ai+b}{c} \rfloor^2$: 当 $a \geq c$ or $b \geq c$ 时, $h(a, b, c, n) = (\frac{a}{c})^2 n(n+1)(2n+1)/6 + (\frac{b}{c})^2 (n+1) + (\frac{a}{c})(\frac{b}{c})n(n+1) + h(a \bmod c, b \bmod c, c, n) + 2(\frac{a}{c})g(a \bmod c, b \bmod c, c, n) + 2(\frac{b}{c})f(a \bmod c, b \bmod c, c, n)$; 否则 $h(a, b, c, n) = nm(m+1) - 2g(c, c-b-1, a, m-1) - 2f(c, c-b-1, a, m-1) - f(a, b, c, n)$ 。

1.5 斯特灵数

- 第一类斯特灵数: 绝对值是 n 个元素划分为 k 个环排列的方案数。 $s(n, k) = s(n-1, k-1) + (n-1)s(n-1, k)$
- 第二类斯特灵数: n 个元素划分为 k 个等价类的方案数。 $S(n, k) = S(n-1, k-1) + kS(n-1, k)$

1.6 一些数论公式

- 当 $x \geq \phi(p)$ 时有 $a^x \equiv a^{x \bmod \phi(p) + \phi(p)} \pmod{p}$
- $\mu^2(n) = \sum_{d^2|n} \mu(d)$
- $\sum_{d|n} \varphi(d) = n$
- $\sum_{d|n} 2^{\omega(d)} = \sigma_0(n^2)$, 其中 ω 是不同素因子个数
- $\sum_{d|n} \mu^2(d) = 2^{\omega(d)}$

1.7 一些数论函数求和的例子

- $\sum_{i=1}^n i[\gcd(i, n) = 1] = \frac{n\varphi(n) + [n=1]}{2}$
- $\sum_{i=1}^n \sum_{j=1}^m [\gcd(i, j) = x] = \sum_d \mu(d) \lfloor \frac{n}{dx} \rfloor \lfloor \frac{m}{dx} \rfloor$
- $\sum_{i=1}^n \sum_{j=1}^m \gcd(i, j) = \sum_{i=1}^n \sum_{j=1}^m \sum_{d|\gcd(i, j)} \varphi(d) = \sum_d \varphi(d) \lfloor \frac{n}{d} \rfloor \lfloor \frac{m}{d} \rfloor$
- $S(n) = \sum_{i=1}^n \mu(i) = 1 - \sum_{i=1}^n \sum_{d|i, d < i} \mu(d) \stackrel{t=\frac{i}{d}}{=} 1 - \sum_{i=2}^n S(\lfloor \frac{n}{i} \rfloor)$ (利用 $[n=1] = \sum_{d|n} \mu(d)$)
- $S(n) = \sum_{i=1}^n \varphi(i) = \sum_{i=1}^n i - \sum_{i=1}^n \sum_{i=1}^n \sum_{d|i, d < i} \varphi(i) \stackrel{t=\frac{i}{d}}{=} \frac{i(i+1)}{2} - \sum_{i=2}^n S(\lfloor \frac{n}{i} \rfloor)$ (利用 $n = \sum_{d|n} \varphi(d)$)
- $\sum_{i=1}^n \mu^2(i) = \sum_{i=1}^n \sum_{d^2|i} \mu(d) = \sum_{d=1}^{\lfloor \sqrt{n} \rfloor} \mu(d) \lfloor \frac{n}{d^2} \rfloor$
- $\sum_{i=1}^n \sum_{j=1}^n \gcd^2(i, j) = \sum_d d^2 \sum_t \mu(t) \lfloor \frac{n}{dt} \rfloor^2$
 $\stackrel{x=dt}{=} \sum_x \lfloor \frac{n}{x} \rfloor^2 \sum_{d|x} d^2 \mu(\frac{x}{d})$
- $\sum_{i=1}^n \varphi(i) = \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n [i \perp j] - 1 = \frac{1}{2} \sum_{i=1}^n \mu(i) \cdot \lfloor \frac{n}{i} \rfloor^2 - 1$

1.8 斐波那契数列性质

- $F_{a+b} = F_{a-1} \cdot F_b + F_a \cdot F_{b+1}$
- $F_1 + F_3 + \dots + F_{2n-1} = F_{2n}, F_2 + F_4 + \dots + F_{2n} = F_{2n+1} - 1$
- $\sum_{i=1}^n F_i = F_{n+2} - 1$
- $\sum_{i=1}^n F_i^2 = F_n \cdot F_{n+1}$
- $F_n^2 = (-1)^{n-1} + F_{n-1} \cdot F_{n+1}$
- $\gcd(F_a, F_b) = F_{\gcd(a, b)}$
- 模 n 周期 (皮萨诺周期)
 - $\pi(p^k) = p^{k-1} \pi(p)$
 - $\pi(mn) = lcm(\pi(n), \pi(m)), \forall n \perp m$
 - $\pi(2) = 3, \pi(5) = 20$
 - $\forall p \equiv \pm 1 \pmod{10}, \pi(p) | p-1$
 - $\forall p \equiv \pm 2 \pmod{5}, \pi(p) | 2p+2$

1.9 常见生成函数

- $(1+ax)^n = \sum_{k=0}^n \binom{n}{k} a^k x^k$
- $\frac{1-x^{r+1}}{1-x^{r+1}} = \sum_{k=0}^n x^k$
- $\frac{1-x}{1-ax} = \sum_{k=0}^{\infty} a^k x^k$

- $\frac{1}{(1-x)^2} = \sum_{k=0}^{\infty} (k+1)x^k$
- $\frac{1}{(1-x)^n} = \sum_{k=0}^{\infty} \binom{n+k-1}{k} x^k$
- $e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$
- $\ln(1+x) = \sum_{k=0}^{\infty} \frac{(-1)^{k+1}}{k} x^k$

1.10 佩尔方程

若一个丢番图方程具有以下形式： $x^2 - ny^2 = 1$ 。且 n 为正整数，则称此二元二次不定方程为**佩尔方程**。

若 n 是完全平方数，则这个方程式只有平凡解 $(\pm 1, 0)$ (实际上对任意的 n , $(\pm 1, 0)$ 都是解)。对于其余情况，拉格朗日证明了佩尔方程总有非平凡解。而这些解可由 \sqrt{n} 的连分数求出。

$$x = [a_0; a_1, a_2, a_3] = x = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \dots}}}$$

设 $\frac{p_i}{q_i}$ 是 \sqrt{n} 的连分数表示： $[a_0; a_1, a_2, a_3, \dots]$ 的渐近分数列，由连分数理论知存在 i 使得 (p_i, q_i) 为佩尔方程的解。取其中最小的 i ，将对应的 (p_i, q_i) 称为佩尔方程的基本解，或最小解，记作 (x_1, y_1) ，则所有的解 (x_i, y_i) 可表示成如下形式： $x_i + y_i\sqrt{n} = (x_1 + y_1\sqrt{n})^i$ 。或者由以下的递回关系式得到：

$$x_{i+1} = x_1x_i + ny_1y_i, \quad y_{i+1} = x_1y_i + y_1x_i。$$

通常，佩尔方程结果的形式通常是 $a_n = ka_{n-1} - a_{n-2}$ (a_{n-2} 前的系数通常是 -1)。暴力 / 凑出两个基础解之后加上一个 0，容易解出 k 并验证。

1.11 Burnside & Polya

$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$ 。 X^g 是 g 下的不动点数量，也就是说有多少种东西用 g 作用之后可以保持不变。

$|X^X/G| = \frac{1}{|G|} \sum_{g \in G} m^{c(g)}$ 。用 m 种颜色染色，然后对于某一种置换 g ，有 $c(g)$ 个置换环，为了保证置换后颜色仍然相同，每个置换环必须染成同色。

1.12 皮克定理

$$2S = 2a + b - 2$$

- S 多边形面积
- a 多边形内部点数
- b 多边形边上点数

1.13 莫比乌斯反演

- $g(n) = \sum_{d|n} f(d) \Leftrightarrow f(n) = \sum_{d|n} \mu(d)g(\frac{n}{d})$
- $f(n) = \sum_{n|d} g(d) \Leftrightarrow g(n) = \sum_{n|d} \mu(\frac{d}{n})f(d)$

1.14 低阶等幂求和

- $\sum_{i=1}^n i^1 = \frac{n(n+1)}{2} = \frac{1}{2}n^2 + \frac{1}{2}n$
- $\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n$

- $\sum_{i=1}^n i^3 = \left[\frac{n(n+1)}{2} \right]^2 = \frac{1}{4}n^4 + \frac{1}{2}n^3 + \frac{1}{4}n^2$
- $\sum_{i=1}^n i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} = \frac{1}{5}n^5 + \frac{1}{2}n^4 + \frac{1}{3}n^3 - \frac{1}{30}n$
- $\sum_{i=1}^n i^5 = \frac{n^2(n+1)^2(2n^2+2n-1)}{12} = \frac{1}{6}n^6 + \frac{1}{2}n^5 + \frac{5}{12}n^4 - \frac{1}{12}n^2$

1.15 一些组合公式

- 错排公式： $D_1 = 0, D_2 = 1, D_n = (n-1)(D_{n-1} + D_{n-2}) = n! (\frac{1}{2!} - \frac{1}{3!} + \dots + (-1)^n \frac{1}{n!}) = \lfloor \frac{n!}{e} + 0.5 \rfloor$
- 卡特兰数 (n 对括号合法方案数, n 个结点二叉树个数, $n \times n$ 方格中对角线下方的单调路径数, 凸 $n+2$ 边形的三角形划分数, n 个元素的合法出栈序列数) : $C_n = \frac{1}{n+1} \binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$

1.16 伯努利数与等幂求和

$\sum_{i=0}^n i^k = \frac{1}{k+1} \sum_{i=0}^k \binom{k+1}{i} B_{k+1-i} (n+1)^i$ 。也可以 $\sum_{i=0}^n i^k = \frac{1}{k+1} \sum_{i=0}^k \binom{k+1}{i} B_{k+1-i} n^i$ 。区别在于 $B_1^+ = 1/2$ 。

1.17 数论分块

$$f(i) = \lfloor \frac{n}{i} \rfloor = v \text{ 时 } i \text{ 的取值范围是 } [l, r]。$$

```
for (LL l = 1, v, r; l <= N; l = r + 1) {
    v = N / l; r = N / v;
}
```

1.18 博弈

- Nim 游戏：每轮从若干堆石子中的一堆取走若干颗。先手必胜条件为石子数量异或非零。
- 阶梯 Nim 游戏：可以选择阶梯上某一堆中的若干颗向下推动一级，直到全部推下去。先手必胜条件是奇数阶梯的异或非零（对于偶数阶梯的操作可以模仿）。
- Anti-SG：无法操作者胜。先手必胜的条件是：
 - SG 不为 0 且某个单—游戏的 SG 大于 1。
 - SG 为 0 且没有单—游戏的 SG 大于 1。
- Every-SG：对所有单—游戏都要操作。先手必胜的条件是单—游戏中的最大 step 为奇数。
 - 对于终止状态 step 为 0
 - 对于 SG 为 0 的状态，step 是最大后继 step + 1
 - 对于 SG 非 0 的状态，step 是最小后继 step + 1
- 树上删边：叶子 SG 为 0，非叶子结点为所有子结点的 SG 值加 1 后的异或和。
- 尝试：
 - 打表找规律
 - 寻找一类必胜态（如对称局面）
 - 直接博弈 dp

2 图论

2.1 带下界网络流

- 无源汇： $u \rightarrow v$ 边容量为 $[l, r]$ ，连容量 $r - l$ ，虚拟源点到 v 连 l ， u 到虚拟汇点连 l 。
 - 有源汇：为了让流能循环使用，连 $T \rightarrow S$ ，容量 ∞ 。
 - 最大流：跑完可行流后，加 $S' \rightarrow S$ ， $T \rightarrow T'$ ，最大流就是答案 ($T \rightarrow S$ 的流量自动退回去了，这一部分就是下界部分的流量)。
 - 最小流： T 到 S 的那条边的实际流量，减去删掉那条边后 T 到 S 的最大流。
 - 网上说可能会减成负的，还要有限地供应 S 之后，再跑一遍 S 到 T 的。
 - 费用流：必要的部分（下界以下的）不要钱，剩下的按照最大流。
- ### 2.2 二分图匹配
- 最小覆盖数 = 最大匹配数
 - 最大独立集 = 顶点数 - 二分图匹配数
 - DAG 最小路径覆盖数 = 结点数 - 拆点后二分图最大匹配数

2.3 差分约束

一个系统 n 个变量和 m 个约束条件组成，每个约束条件形如 $x_j - x_i \leq b_k$ 。可以发现每个约束条件都形如最短路中的三角不等式 $d_u - d_v \leq w_{u,v}$ 。因此连一条边 (i, j, b_k) 建图。

若要使得所有量两两的值最接近，源点到各点的距离初始成 0，跑最短路。

若要使得某一变量与其他变量的差尽可能大，则源点到各点距离初始化成 ∞ ，跑最短路。

2.4 三元环

将点分成度入小于 \sqrt{m} 和超过 \sqrt{m} 的两类。现求包含第一类点的三元环个数。由于边数较少，直接枚举两条边即可。由于一个点度数不超过 \sqrt{m} ，所以一条边最多被枚举 \sqrt{m} 次，复杂度 $O(m\sqrt{m})$ 。再求不包含第一类点的三元环个数，由于这样的点不超过 \sqrt{m} 个，所以复杂度也是 $O(m\sqrt{m})$ 。

对于每条无向边 (u, v) ，如果 $d_u < d_v$ ，那么连有向边 (u, v) ，否则有向边 (v, u) 。度数相等的按第二关键字判断。然后枚举每个点 x ，假设 x 是三元组中度数最小的点，然后暴力往后面枚举两条边找到 y ，判断 (x, y) 是否有边即可。复杂度也是 $O(m\sqrt{m})$ 。

2.5 四元环

考虑这样一个四元环，将答案统计在度数最大的点 b 上。考虑枚举点 u ，然后枚举与其相邻的点 v ，然后再枚举所有度数比 v 大的与 v 相邻的点，这些点显然都可能作为 b 点，我们维护一个计数器来计算之前 b 被枚举多少次，答案加上计数器的值，然后计数器加一。

枚举完 u 之后，我们用和枚举时一样的方法来清空计数器就好了。

任何一个点，与其直接相连的度数大于等于它的点最多只有 $\sqrt{2m}$ 个。所以复杂度 $O(m\sqrt{m})$ 。

2.6 支配树

- semi[x]** 必经点 (就是 x 的祖先 z 中，能不经过 z 和 x 之间的树上的点而到达 x 的点中深度最小的)
- idom[x]** 最近必经点 (就是深度最大的根到 x 的必经点)

3 计算几何

3.1 k 次圆覆盖

一种是用竖线进行切分，然后对每一个切片分别计算。扫描线部分可以魔改，求各种东西。复杂度 $O(n^3 \log n)$ 。

复杂度 $O(n^2 \log n)$ 。原理是：认为所求部分是一个奇怪的多边形 + 若干弓形。然后对于每个圆分别求贡献的弓形，并累加多边形有向面积。可以魔改扫描线的部分，用于求周长、至少覆盖 k 次等等。内含、内切、同一个圆的情况，通常需要特殊处理。

3.2 三维凸包

增量法。先将所有的点打乱顺序，然后选择四个不共面的点组成一个四面体，如果找不到说明凸包不存在。然后遍历剩余的点，不断更新凸包。对遍历到的点做如下处理。

- 如果点在凸包内，则不更新。
- 如果点在凸包外，那么找到所有原凸包上所有分隔了这个点可见面和不可见面的边，以这样的边的两个点和新点的点创建新的面加入凸包中。

4 随机素数表

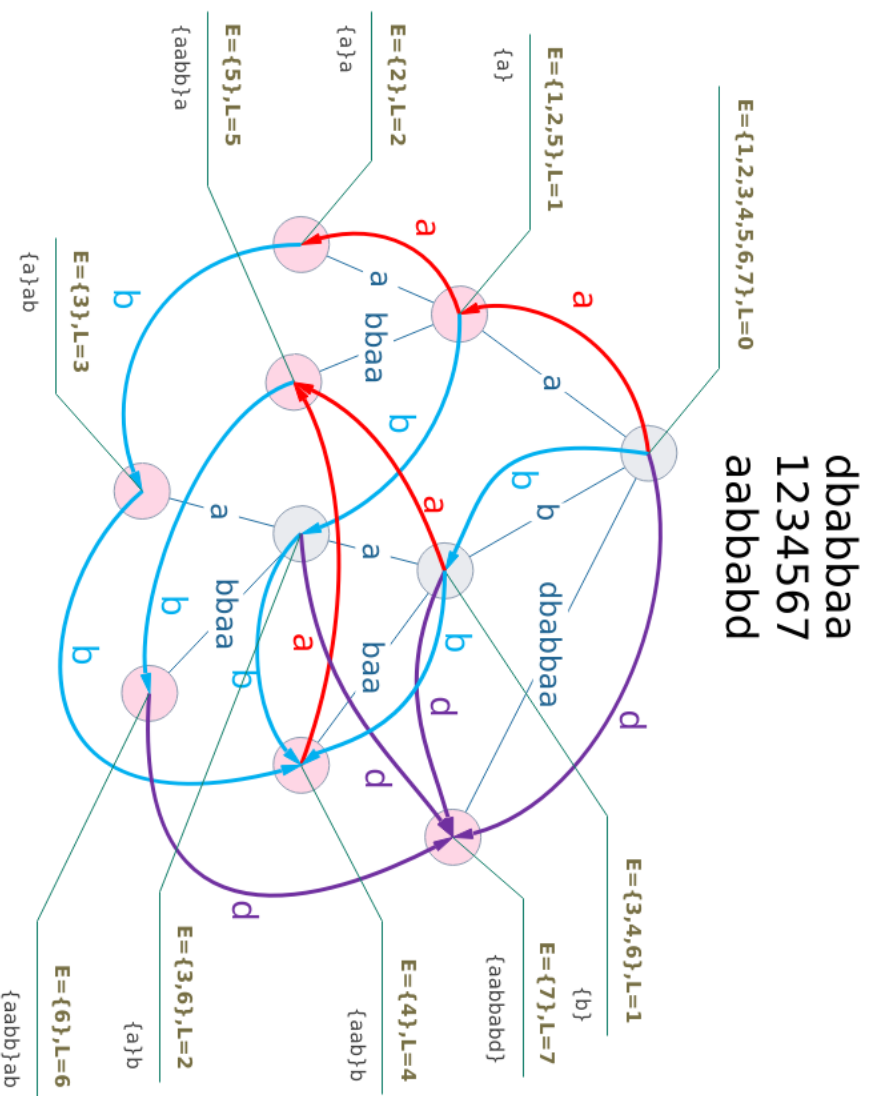
42737, 46411, 50101, 52627, 54577, 191677, 194869, 210407, 221831, 241337, 578603, 625409, 713569, 788813, 862481, 2174729, 2326673, 2688877, 2779417, 3133583, 4489747, 6697841, 6791471, 6878533, 7883129, 9124553, 10415371, 11134633, 12214801, 15589333, 17148757, 17997457, 20278487, 27256133, 28678757, 38206199, 41337119, 47422547, 48543479, 52834961, 76993291, 85852231, 95217823, 108755563, 132972461, 171863609, 173629837, 176939899, 207808351, 227218703, 306112619, 311809637, 322711981, 330806107, 345593317, 345887293, 362838523, 373523729, 394207349, 409580177, 437359931, 483577261, 490845269, 512059357, 534387017, 698987533, 764016151, 906097321, 914067307, 954169327

适合哈希的素数：1572869, 3145739, 6291469, 12582917, 25165843, 50331653

NTT 素数表: $p = r \cdot 2^k + 1$, 原根是 g . 3, 1, 1, 2; 5, 1, 2, 2; 17, 1, 4, 3; 97, 3, 5, 5; 193, 3, 6, 5; 257, 1, 8, 3; 7681, 15, 9, 17; 12289, 3, 12, 11; 40961, 5, 13, 3; 65537, 1, 16, 3; 786433, 3, 18, 10; 5767169, 11, 19, 3; 7340033, 7, 20, 3; 23068673, 11, 21, 3; 104857601, 25, 22, 3; 167772161, 5, 25, 3; 469762049, 7, 26, 3; 1004535809, 479, 21, 3; 2013265921, 15, 27, 31; 2281701377, 17, 27, 3; 3221225473, 3, 30, 5; 75161927681, 35, 31, 3; 77309411329, 9, 33, 7; 206158430209, 3, 36, 22; 2061584302081, 15, 37, 7; 2748779069441, 5, 39, 3; 6597069766657, 3, 41, 5; 3958241859937, 9, 42, 5; 79164837199873, 9, 43, 5; 263882790666241, 15, 44, 7; 1231453023109121, 35, 45, 3; 1337006139375617, 19, 46, 3; 3799912185593857, 27, 47, 5.

5 心态崩了

- `(int)v.size()`
- `1LL << k`
- 递归函数用全局或者 static 变量要小心
- 预处理组合数注意上限
- 想清楚到底是要 `multiset` 还是 `set`
- 提交之前看一下数据范围, 测一下边界



- 数据结构注意数组大小 (2 倍, 4 倍)
- 字符串注意字符集
- 如果函数中使用了默认参数的话, 注意调用时的参数个数
- 注意要读完
- 构造参数无法使用自己
- 树链剖分/dfs 序, 初始化或者询问不要忘记 `idx`, `ridx`
- 排序时注意结构体的所有属性是不是考虑了
- 不要把 `while` 写成 `if`
- 不要把 `int` 开成 `char`
- 清零的时候全部用 0 到 $n + 1$ 。
- 模意义下不要用除法
- 哈希不要自然溢出
- 最短路不要 SPFA, 乖乖写 Dijkstra
- 上取整以及 GCD 小心负数
- `mid` 用 `1 + (r - 1) / 2` 可以避免溢出和负数的问题
- 小心模板自带的意料之外的隐式类型转换
- 求最优解时不要忘记更新当前最优解
- 图论问题一定要注意图不连通的问题
- 处理强制在线的时候 `lastans` 负数也要记得矫正
- 不要觉得编译器什么都能优化