

Contents

1 Misc	1	4.2.1 Matroid Intersection	14
1.1 Contest	1	4.2.2 De Bruijn Sequence	15
1.1.1 Makefile	1	4.2.3 Multinomial	15
1.2 How Did We Get Here?	1	4.3 Algebra	15
1.2.1 Macros	1	4.3.1 Formal Power Series	15
1.2.2 Fast I/O	2	4.4 Theorems	16
1.2.3 constexpr	2	4.4.1 Kirchhoff's Theorem	16
1.2.4 Bump Allocator	2	4.4.2 Tutte's Matrix	16
1.3 Tools	2	4.4.3 Cayley's Formula	16
1.3.1 Floating Point Binary Search	2	4.4.4 Erdős-Gallai Theorem	16
1.3.2 SplitMix64	2	4.4.5 Burnside's Lemma	16
1.3.3 <random>	3		
1.3.4 x86 Stack Hack	3		
1.4 Algorithms	3	5 Numeric	16
1.4.1 Bit Hacks	3	5.1 Barrett Reduction	16
1.4.2 Aliens Trick	3	5.2 Long Long Multiplication	16
1.4.3 Hilbert Curve	3	5.3 Fast Fourier Transform	17
1.4.4 Infinite Grid Knight Distance	3	5.4 Fast Walsh-Hadamard Transform	17
1.4.5 Poker Hand	3	5.5 Subset Convolution	17
1.4.6 Longest Increasing Subsequence	3	5.6 Linear Recurrences	17
1.4.7 Mo's Algorithm on Tree	3	5.6.1 Berlekamp-Massey Algorithm	17
2 Data Structures	4	5.6.2 Linear Recurrence Calculation	17
2.1 GNU PBDS	4	5.7 Matrices	17
2.2 Segment Tree (ZKW)	4	5.7.1 Determinant	17
2.3 Line Container	4	5.7.2 Inverse	18
2.4 Li-Chao Tree	4	5.7.3 Characteristic Polynomial	18
2.5 Heavy-Light Decomposition	5	5.7.4 Solve Linear Equation	18
2.6 Wavelet Matrix	5	5.8 Polynomial Interpolation	19
2.7 Link-Cut Tree	5	5.9 Simplex Algorithm	19
3 Graph	6		
3.1 Modeling	6	6 Geometry	19
3.2 Matching/Flows	7	6.1 Point	19
3.2.1 Dinic's Algorithm	7	6.1.1 Quaternion	20
3.2.2 Minimum Cost Flow	7	6.1.2 Spherical Coordinates	20
3.2.3 Gomory-Hu Tree	7	6.2 Segments	20
3.2.4 Global Minimum Cut	7	6.3 Convex Hull	20
3.2.5 Bipartite Minimum Cover	8	6.3.1 3D Hull	20
3.2.6 Edmonds' Algorithm	8	6.4 Angular Sort	21
3.2.7 Minimum Weight Matching	8	6.5 Convex Polygon Minkowski Sum	21
3.2.8 Stable Marriage	9	6.6 Point In Polygon	21
3.2.9 Kuhn-Munkres algorithm	9	6.6.1 Convex Version	21
3.3 Shortest Path Faster Algorithm	10	6.6.2 Offline Multiple Points Version	21
3.4 Strongly Connected Components	10	6.7 Closest Pair	22
3.4.1 2-Satisfiability	10	6.8 Minimum Enclosing Circle	22
3.5 Biconnected Components	10	6.9 Delaunay Triangulation	22
3.5.1 Articulation Points	10	6.9.1 Slower Version	23
3.5.2 Bridges	10	6.10 Half Plane Intersection	23
3.6 Triconnected Components	11		
3.7 Centroid Decomposition	11		
3.8 Minimum Mean Cycle	11		
3.9 Directed MST	11		
3.10 Maximum Clique	12		
3.11 Dominator Tree	12		
3.12 Manhattan Distance MST	13		
4 Math	13		
4.1 Number Theory	13	7 Strings	23
4.1.1 Mod Struct	13	7.1 Knuth-Morris-Pratt Algorithm	23
4.1.2 Miller-Rabin	13	7.2 Aho-Corasick Automaton	23
4.1.3 Linear Sieve	13	7.3 Suffix Array	24
4.1.4 Get Factors	13	7.4 Suffix Tree	24
4.1.5 Binary GCD	13	7.5 Cocke-Younger-Kasami Algorithm	24
4.1.6 Extended GCD	13	7.6 Z Value	25
4.1.7 Chinese Remainder Theorem	13	7.7 Manacher's Algorithm	25
4.1.8 Baby-Step Giant-Step	13	7.8 Minimum Rotation	25
4.1.9 Pollard's Rho	14	7.9 Palindromic Tree	25
4.1.10 Tonelli-Shanks Algorithm	14		
4.1.11 Chinese Sieve	14		
4.1.12 Rational Number Binary Search	14		
4.1.13 Farey Sequence	14		
4.2 Combinatorics	14		

1. Misc

1.1. Contest

1.1.1. Makefile

```

1 .PRECIOUS: ./p%
3 %: p%
4   ulimit -s unlimited && ./<
5 p%: p%.cpp
6   g++ -o $@ $< -std=c++17 -Wall -Wextra -Wshadow \
7     -fsanitize=address,undefined

```

1.2. How Did We Get Here?

1.2.1. Macros

Use vectorizations and math optimizations at your own peril.
For $\text{gcc} \geq 9$, there are `[[likely]]` and `[[unlikely]]` attributes.

Call gcc with `-fopt-info-optimized-missed-optall` for optimization info.

```

1 #define _GLIBCXX_DEBUG           1 // for debug mode
#define _GLIBCXX_SANITIZE_VECTOR 1 // for asan on vectors
3 #pragma GCC optimize("O3", "unroll-loops")
# pragma GCC optimize("fast-math")
# pragma GCC target("avx,avx2,abm,bmi,bmi2") // tip: `lscpu` 
// before a loop
7 #pragma GCC unroll 16 // 0 or 1 -> no unrolling
# pragma GCC ivdep

```

1.2.2. Fast I/O

```

1 struct scanner {
2     static constexpr size_t LEN = 32 << 20;
3     char *buf, *buf_ptr, *buf_end;
4     scanner()
5         : buf(new char[LEN]), buf_ptr(buf + LEN),
6             buf_end(buf + LEN) {}
7     ~scanner() { delete[] buf; }
8     char getc() {
9         if (buf_ptr == buf_end) [[unlikely]]
10            buf_end = buf + fread_unlocked(buf, 1, LEN, stdin),
11            buf_ptr = buf;
12        return *(buf_ptr++);
13    }
14    char seek(char del) {
15        char c;
16        while ((c = getc()) < del) {}
17        return c;
18    }
19    void read(int &t) {
20        bool neg = false;
21        char c = seek('-');
22        if (c == '-') neg = true, t = 0;
23        else t = c ^ '0';
24        while ((c = getc()) >= '0') t = t * 10 + (c ^ '0');
25        if (neg) t = -t;
26    }
27    struct printer {
28        static constexpr size_t CPI = 21, LEN = 32 << 20;
29        char *buf, *buf_ptr, *buf_end, *tbuf;
30        char *int_buf, *int_buf_end;
31        printer()
32            : buf(new char[LEN]), buf_ptr(buf),
33                buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
34                int_buf_end(int_buf + CPI - 1) {}
35        ~printer() {
36            flush();
37            delete[] buf, delete[] int_buf;
38        }
39        void flush() {
40            fwrite_unlocked(buf, 1, buf_ptr - buf, stdout);
41            buf_ptr = buf;
42        }
43        void write_(const char &c) {
44            *buf_ptr = c;
45            if (++buf_ptr == buf_end) [[unlikely]]
46                flush();
47        }
48        void write_(const char *s) {
49            for (; *s != '\0'; ++s) write_(*s);
50        }
51        void write(int x) {
52            if (x < 0) write_('-'), x = -x;
53            if (x == 0) [[unlikely]]
54                return write_('0');
55            for (tbuf = int_buf_end; x != 0; --tbuf, x /= 10)
56                *tbuf = '0' + char(x % 10);
57            write_(++tbuf);
58        }
59    };

```

Kotlin

```

1 import java.io.*
2 import java.util.*
3
4 @JvmField val cin = System.`in`.bufferedReader()
5 @JvmField val cout = PrintWriter(System.out, false)
6 @JvmField var tokenizer: StringTokenizer = StringTokenizer("")
7 fun nextLine(): String =
8     while (!tokenizer.hasMoreTokens())
9         tokenizer = StringTokenizer(nextLine())
10    return tokenizer.nextToken()
11
12 // example
13 fun main() {
14
15

```

```

17     val n = read().toInt()
18     val a = DoubleArray(n) { read().toDouble() }
19     cout.println("omg hi")
20     cout.flush()
21 }

```

1.2.3. constexpr

Some default limits in gcc (7.x - trunk):

- `constexpr` recursion depth: 512
- `constexpr` loop iteration per function: 262144
- `constexpr` operation count per function: 33554432
- template recursion depth: 900 (gcc might segfault first)

```

1 constexpr array<int, 10> fibonacci[] {
2     array<int, 10> a{};
3     a[0] = a[1] = 1;
4     for (int i = 2; i < 10; i++) a[i] = a[i - 1] + a[i - 2];
5     return a;
6 }();
7 static_assert(fibonacci[9] == 55, "CE");
8
9 template <typename F, typename INT, INT... S>
10 constexpr void for_constexpr(integer_sequence<INT, S...>, 
11                             F &&func) {
12     int _[] = {{func(integral_constant<INT, S>{}), 0}...};
13 }
14 // example
15 template <typename... T> void print_tuple(tuple<T...> t) {
16     for_constexpr(make_index_sequence<sizeof...(T)>{}, 
17                  [&](auto i) { cout << get<i>(t) << '\n'; });
18 }

```

1.2.4. Bump Allocator

```

1
2
3 // global bump allocator
4 char mem[256 << 20]; // 256 MB
5 size_t rsp = sizeof mem;
6 void *operator new(size_t s) {
7     assert(s < rsp); // MLE
8     return (void *)&mem[rsp -= s];
9 }
10 void operator delete(void *) {}
11
12 // bump allocator for STL / pbds containers
13 char mem[256 << 20];
14 size_t rsp = sizeof mem;
15 template <typename T> struct bump {
16     typedef T value_type;
17     bump() {}
18     template <typename U> bump(U, ...) {}
19     T *allocate(size_t n) {
20         rsp -= n * sizeof(T);
21         rsp &= 0 - alignof(T);
22         return (T*)(mem + rsp);
23     }
24     void deallocate(T *, size_t n) {}
25 };

```

1.3. Tools

1.3.1. Floating Point Binary Search

```

1 union di {
2     double d;
3     ull i;
4 };
5 bool check(double);
6 // binary search in [L, R] with relative error 2^-eps
7 double binary_search(double L, double R, int eps) {
8     di l = {L}, r = {R}, m;
9     while (r.i - l.i > 1LL << (52 - eps)) {
10         m.i = (l.i + r.i) >> 1;
11         if (check(m.d)) r = m;
12         else l = m;
13     }
14     return l.d;
15 }

```

1.3.2. SplitMix64

```

1 using ull = unsigned long long;
2 inline ull splitmix64(ull x) {
3     // change to `static ull x = SEED;` for DRBG
4     ull z = (x += 0x9E3779B97F4A7C15);
5     z = (z ^ (z >> 30)) * 0xBF58476D1CE4E5B9;
6     z = (z ^ (z >> 27)) * 0x94D049BB133111EB;
7     return z ^ (z >> 31);
8 }

```

1.3.3. <random>

```

1 #ifdef __unix__
2     random_device rd;
3     mt19937_64 RNG(rd());
4 #else
5     const auto SEED = chrono::high_resolution_clock::now()
6                         .time_since_epoch()
7                         .count();
8     mt19937_64 RNG(SEED);
9 #endif
10 // random uint_fast64_t: RNG();
11 // uniform random of type T (int, double, ...) in [l, r];
12 // uniform_int_distribution<T> dist(l, r); dist(RNG);

```

1.3.4. x86 Stack Hack

```
1 constexpr size_t size = 200 << 20; // 200MiB
2 int main() {
3     register long rsp asm("rsp");
4     char *buf = new char[size];
5     asm("movq %0, %%rsp\n" ::"r"(buf + size));
6     // do stuff
7     asm("movq %0, %%rsp\n" ::"r"(rsp));
8     delete[] buf;
9 }
```

1.4. Algorithms

1.4.1. Bit Hacks

```
1 // next permutation of x as a bit sequence
2 ull next_bits_permutation(ull x) {
3     ull c = __builtin_ctzll(x), r = x + (1ULL << c);
4     return (r ^ x) >> (c + 2) | r;
5 }
6 // iterate over all (proper) subsets of bitset s
7 void subsets(ull s) {
8     for (ull x = s; x;) { --x &= s; /* do stuff */ }
9 }
```

1.4.2. Aliens Trick

```

1 // min dp[i] value and its i (smallest one)
2 pll get_dp(int cost);
3 ll aliens(int k, int l, int r) {
4     while (l != r) {
5         int m = (l + r) / 2;
6         auto [f, s] = get_dp(m);
7         if (s == k) return f - m * k;
8         if (s < k) r = m;
9         else l = m + 1;
10    }
11    return get_dp(l).first - l * k;
}

```

1.4.3. Hilbert Curve

```
1 ll hilbert(ll n, int x, int y) {
2     ll res = 0;
3     for (ll s = n; s /= 2;) {
4         int rx = !(x & s), ry = !(y & s);
5         res += s * s * ((3 * rx) ^ ry);
6         if (ry == 0) {
7             if (rx == 1) x = s - 1 - x, y = s - 1 - y;
8             swap(x, y);
9         }
10    }
11    return res;
}
```

1.4.4. Infinite Grid Knight Distance

```
1 ll get_dist(ll dx, ll dy) {
2     if (++(dx = abs(dx)) > ++(dy = abs(dy))) swap(dx, dy);
3     if (dx == 1 && dy == 2) return 3;
4     if (dx == 3 && dy == 3) return 4;
5     ll lb = max(dy / 2, (dx + dy) / 3);
6     return ((dx ^ dy ^ lb) & 1) ? ++lb : lb;
7 }
```

1.4.5. Poker Hand

```
1
2
3
4
5
6
7 using namespace std;
8
9 struct hand {
10     static constexpr auto rk = [ ] {
```

```

11 array<int, 256> x{};

12 auto s = "23456789TJQKACDHS";
13 for (int i = 0; i < 17; i++) x[s[i]] = i % 13;
14 return x;
15 }

16 vector<pair<int, int>> v;
17 vector<int> cnt, vf, vs;
18 int type;
19 hand() : cnt(4), type(0) {}
20 void add_card(char suit, char rank) {
21     ++cnt[rk[suit]];
22     for (auto &[f, s] : v)
23         if (s == rk[rank]) return ++f, void();
24     v.emplace_back(1, rk[rank]);
25 }
26 void process() {
27     sort(v.rbegin(), v.rend());
28     for (auto [f, s] : v) vf.push_back(f), vs.push_back(s);
29     bool str = 0, flu = find(all(cnt), 5) != cnt.end();
30     if ((str = v.size() == 5))
31         for (int i = 1; i < 5; i++)
32             if (vs[i] != vs[i - 1] + 1) str = 0;
33     if (vs == vector<int>{12, 3, 2, 1, 0})
34         str = 1, vs = {3, 2, 1, 0, -1};
35     if (str && flu) type = 9;
36     else if (vf[0] == 4) type = 8;
37     else if (vf[0] == 3 && vf[1] == 2) type = 7;
38     else if (str || flu) type = 5 + flu;
39     else if (vf[0] == 3) type = 4;
40     else if (vf[0] == 2) type = 2 + (vf[1] == 2);
41     else type = 1;
42 }
43 bool operator<(const hand &b) const {
44     return make_tuple(type, vf, vs) <
45             make_tuple(b.type, b.vf, b.vs);
46 }

```

1.4.6. Longest Increasing Subsequence

```

1
3 template <class I> vi lis(const vector<I> &S) {
4     if (S.empty()) return {};
5     vi prev(sz(S));
6     ttypedef pair<I, int> p;
7     vector<p> res;
8     rep(i, 0, sz(S)) {
9         // change 0 -> i for longest non-decreasing subsequence
10        auto it = lower_bound(all(res), p{S[i], 0});
11        if (it == res.end())
12            res.emplace_back(), it = res.end() - 1;
13        *it = {S[i], i};
14        prev[i] = it == res.begin() ? 0 : (it - 1)->second;
15    }
16    int L = sz(res), cur = res.back().second;
17    vi ans(L);
18    while (L--) ans[L] = cur, cur = prev[cur];
19    return ans;
}

```

1.4.7. Mo's Algorithm on Tree

```

1 void MoAlgoOnTree() {
2     Dfs(0, -1);
3     vector<int> euler(tk);
4     for (int i = 0; i < n; ++i) {
5         euler[tin[i]] = i;
6         euler[tout[i]] = i;
7     }
8     vector<int> l(q), r(q), qr(q), sp(q, -1);
9     for (int i = 0; i < q; ++i) {
10        if (tin[u[i]] > tin[v[i]]) swap(u[i], v[i]);
11        int z = GetLCA(u[i], v[i]);
12        sp[i] = z[i];
13        if (z == u) l[i] = tin[u[i]], r[i] = tin[v[i]];
14        else l[i] = tout[u[i]], r[i] = tin[v[i]];
15        qr[i] = i;
16    }
17    sort(qr.begin(), qr.end(), [&](int i, int j) {
18        if (l[i] / kB == l[j] / kB) return r[i] < r[j];
19        return l[i] / kB < l[j] / kB;
20    });
21    vector<bool> used(n);
22    // Add(v): add/remove v to/from the path based on used[v]
23    for (int i = 0, tl = 0, tr = -1; i < q; ++i) {
24        while (tl < l[qr[i]]) Add(euler[tl++]);
25        while (tl > l[qr[i]]) Add(euler[--tl]);
26        while (tr > r[qr[i]]) Add(euler[tr--]);
27        while (tr < r[qr[i]]) Add(euler[++tr]);
28        // add/remove LCA(u, v) if necessary
29    }
}

```

2. Data Structures

2.1. GNU PBDS

```

1 #include <ext/pb_ds/assoc_container.hpp>
2 #include <ext/pb_ds/priority_queue.hpp>
3 #include <ext/pb_ds/tree_policy.hpp>
4 using namespace __gnu_pbds;
5
6 // most std::map + order_of_key, find_by_order, split, join
7 template <typename T, typename U = null_type>
8 using ordered_map = tree<T, U, std::less<>, rb_tree_tag,
9                         tree_order_statistics_node_update>;
10 // useful tags: rb_tree_tag, splay_tree_tag
11
12 template <typename T> struct myhash {
13     size_t operator()(T x) const; // splitmix, bswap(x*R), ...
14 };
15 // most of std::unordered_map, but faster (needs good hash)
16 template <typename T, typename U = null_type>
17 using hash_table = gp_hash_table<T, U, myhash<T>>;
18
19 // most std::priority_queue + modify, erase, split, join
20 using heap = priority_queue<int, std::less<>>;
21 // useful tags: pairing_heap_tag, binary_heap_tag,
22 // (rc_)?binomial_heap_tag, thin_heap_tag

```

2.2. Segment Tree (ZKW)

```

1 struct segtree {
2     using T = int;
3     T f(T a, T b) { return a + b; } // any monoid operation
4     static constexpr T ID = 0; // identity element
5     int n;
6     vector<T> v;
7     segtree(int n_) : n(n_), v(2 * n, ID) {}
8     segtree(vector<T> &a) : n(a.size()), v(2 * n, ID) {
9         copy_n(a.begin(), n, v.begin() + n);
10        for (int i = n - 1; i > 0; i--) {
11            v[i] = f(v[i * 2], v[i * 2 + 1]);
12        }
13        void update(int i, T x) {
14            for (v[i += n] = x; i /= 2;) {
15                v[i] = f(v[i * 2], v[i * 2 + 1]);
16            }
17        T query(int l, int r) {
18            T tl = ID, tr = ID;
19            for (l += n, r += n; l < r; l /= 2, r /= 2) {
20                if (l & 1) tl = f(tl, v[l++]);
21                if (r & 1) tr = f(v[--r], tr);
22            }
23            return f(tl, tr);
24        }
25    };

```

2.3. Line Container

```

1
2 struct Line {
3     mutable ll k, m, p;
4     bool operator<(const Line &o) const { return k < o.k; }
5     bool operator<(ll x) const { return p < x; }
6 };
7
8 // add: line y=kx+m, query: maximum y of given x
9 struct LineContainer : multiset<Line, less<>> {
10     // (for doubles, use inf = 1/.0, div(a,b) = a/b)
11     static const ll inf = LLONG_MAX;
12     ll div(ll a, ll b) { // floored division
13         return a / b - ((a ^ b) < 0 && a % b);
14     }
15     bool isect(iterator x, iterator y) {
16         if (y == end()) return x->p = inf, 0;
17         if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
18         else x->p = div(y->m - x->m, x->k - y->k);
19         return x->p >= y->p;
20     }
21     void add(ll k, ll m) {
22         auto z = insert({k, m, 0}), y = z++, x = y;
23         while (isect(y, z)) z = erase(z);
24         if (x != begin() && isect(--x, y))
25             isect(x, y = erase(y));
26         while ((y = x) != begin() && (--x)->p >= y->p)
27             isect(x, erase(y));
28     }
29     ll query(ll x) {
30         assert(!empty());
31         auto l = *lower_bound(x);
32         return l.k * x + l.m;
33     }

```

2.4. Li-Chao Tree

```

1 public class LiChaoTree {
2
3     // Represents a line y = mx + c
4     static class Line {
5         long m, c;
6
7         public Line(long m, long c) {
8             this.m = m;
9             this.c = c;
10        }
11
12        // Evaluates the line at a given x-coordinate
13        long eval(long x) {
14            return m * x + c;
15        }
16    }
17
18    // Node of the Li-Chao Tree
19    static class Node {
20        Line line;
21        Node left, right;
22
23        public Node(Line line) {
24            this.line = line;
25        }
26    }
27
28    private Node root;
29    private final long minCoord;
30    private final long maxCoord;
31    private final Line identityLine; // Represents "no line" or infinity
32
33    // Constructor for the Li-Chao Tree
34    // minCoord and maxCoord define the range of x-values the tree
35    // handles. identityLine should return a very large value for min queries
36    public LiChaoTree(long minCoord, long maxCoord) {
37        this.minCoord = minCoord;
38        this.maxCoord = maxCoord;
39        // For minimum queries, an identity line should return a very large value
40        // Using Long.MAX_VALUE for 'c' and 0 for 'm' ensures it's never chosen
41        this.identityLine = new Line(0, Long.MAX_VALUE);
42        this.root = new Node(identityLine);
43    }
44
45    // Adds a new line to the tree
46    public void addLine(Line newLine) {
47        addLine(root, minCoord, maxCoord, newLine);
48    }
49
50    private void addLine(Node node, long currentMin, long currentMax) {
51        long mid = currentMin + (currentMax - currentMin) / 2;
52        boolean leftBetter = newLine.eval(currentMin) < node.line.eval(currentMin);
53        boolean midBetter = newLine.eval(mid) < node.line.eval(mid);
54
55        if (midBetter) {
56            // If the new line is better at the midpoint, swap it with the old one
57            Line temp = node.line;
58            node.line = newLine;
59            newLine = temp; // The old line now becomes the 'new' one
60        }
61
62        // If the interval is a single point, we are done
63        if (currentMin == currentMax) {
64            return;
65        }
66
67        // Decide which child to push the 'worse' line to
68        if (leftBetter != midBetter) { // Intersection point is in the left half
69            if (node.left == null) {
70                node.left = new Node(identityLine);
71            }
72            addLine(node.left, currentMin, mid, newLine);
73        } else if (leftBetter == midBetter && leftBetter == false) {
74            if (node.right == null) {
75                node.right = new Node(identityLine);
76            }
77            addLine(node.right, mid + 1, currentMax, newLine);
78        }
79
80        // If leftBetter == midBetter == true, it means the new line
81        // and the old line is completely dominated, so no need to do anything
82    }
83
84    // Queries the minimum value at a given x-coordinate
85    public long query(long x) {
86        return query(root, minCoord, maxCoord, x);
87    }
88
89    private long query(Node node, long currentMin, long currentMax) {
90        if (node == null) {
91            return identityLine.eval(x); // No line in this path, return infinity
92        }
93    }

```

```

93    long res = node.line.eval(x);
95    if (currentMin == currentMax) {
96        return res; // Reached a leaf node
97    }
99
100   long mid = currentMin + (currentMax - currentMin) / 2;
101   if (x <= mid) {
102       res = Math.min(res, query(node.left, currentMin, mid));
103   } else {
104       res = Math.min(res, query(node.right, mid + 1, currentMax));
105   }
106   return res;
107 }
108
109 public static void main(String[] args) {
110     // Example Usage:
111     // Create a Li-Chao Tree for x-coordinates from -1000 to 1000
112     LiChaoTree lct = new LiChaoTree(-1000, 1000);
113
114     // Add some lines: y = mx + c
115     lct.addLine(new Line(1, 5)); // y = x + 5
116     lct.addLine(new Line(-1, 10)); // y = -x + 10
117     lct.addLine(new Line(0, 7)); // y = 7
118
119     // Query minimum values at different x-coordinates
120     System.out.println("Min at x = 0: " + lct.query(0));
121     System.out.println("Min at x = 2: " + lct.query(2));
122     System.out.println("Min at x = 6: " + lct.query(6));
123     System.out.println("Min at x = -5: " + lct.query(-5));
124
125     lct.addLine(new Line(2, -3)); // y = 2x - 3
126     System.out.println("Min at x = 0 (after new line): " + lct.query(0));
127     System.out.println("Min at x = 2 (after new line): " + lct.query(2));
128 }

```

2.5. Heavy-Light Decomposition

```

1
2 template <bool VALS_EDGES> struct HLD {
3     int N, tim = 0;
4     vector<vi> adj;
5     vi par, siz, depth, rt, pos;
6     Node *tree;
7     HLD(vector<vi> adj_) : N(sz(adj_)), adj(adj_), par(N, -1), siz(N, 1),
8         depth(N), rt(N), pos(N), tree(new Node(0, N)) {
9         dfsSz(0);
10        dfsHld(0);
11    }
12    void dfsSz(int v) {
13        if (par[v] != -1)
14            adj[v].erase(find(all(adj[v]), par[v]));
15        for (int &u : adj[v]) {
16            par[u] = v, depth[u] = depth[v] + 1;
17            dfsSz(u);
18            siz[v] += siz[u];
19            if (siz[u] > siz[adj[v][0]]) swap(u, adj[v][0]);
20        }
21    }
22    void dfsHld(int v) {
23        pos[v] = tim++;
24        for (int u : adj[v]) {
25            rt[u] = (u == adj[v][0] ? rt[v] : u);
26            dfsHld(u);
27        }
28    }
29
30    template <class B> void process(int u, int v, B op) {
31        for (; rt[u] != rt[v]; v = par[rt[v]]) {
32            if (depth[rt[u]] > depth[rt[v]]) swap(u, v);
33            op(pos[rt[v]], pos[v] + 1);
34        }
35        if (depth[u] > depth[v]) swap(u, v);
36        op(pos[u] + VALS_EDGES, pos[v] + 1);
37    }
38    void modifyPath(int u, int v, int val) {
39        process(u, v,
40                 [&](int l, int r) { tree->add(l, r, val); });
41    }
42    int queryPath(int u,
43                  int v) { // Modify depending on problem
44        int res = -1e9;
45        process(u, v, [&](int l, int r) {
46            res = max(res, tree->query(l, r));
47        });
48        return res;
49    }
50    int querySubtree(int v) { // modifySubtree is similar
51        return tree->query(pos[v] + VALS_EDGES,
52                            pos[v] + siz[v]);
53    }
54}

```

2.6. Wavelet Matrix

```

1
2 #pragma GCC target("popcnt,bmi2")
3 #include <immintrin.h>
4
5 template <typename T> struct wavelet_matrix {
6     static_assert(is_unsigned_v<T>, "only unsigned T");
7     struct bit_vector {
8         static constexpr uint W = 64;
9         uint n, cnt0;
10        vector<ull> bits;
11        vector<uint> sum;
12        bit_vector(uint n_) : n(n_), bits(n / W + 1), sum(n / W + 1) {}
13        void build() {
14            for (uint j = 0; j != n / W; ++j)
15                sum[j + 1] = sum[j] + _mm_popcnt_u64(bits[j]);
16            cnt0 = rank0(n);
17        }
18        void set_bit(uint i) { bits[i / W] |= 1ULL << i % W; }
19        bool operator[](uint i) const {
20            return !(bits[i / W] & 1ULL << i % W);
21        }
22        Expected: min(5, 10, 7) = 5
23        uint rank1(uint i) const {
24            return sum[i / W] >= 4;
25        }
26        Expected: _mm_popcnt_u64(_bzhi_u64(bits[i / W], i % W));
27        uint rank0(uint i) const { return i - rank1(i); }
28        Expected: min(5, 10, 7, -3) = -3
29        Expected: min(7, 8, 7, 1) = 1
30        vector<bit_vector> b;
31        wavelet_matrix(const vector<T> &a) : n(a.size()) {
32            lg =
33                __lg(max(*max_element(a.begin(), a.end()), T(1))) + 1;
34            b.assign(lg, n);
35            vector<T> cur = a, nxt(n);
36            for (int h = lg; h--;) {
37                for (uint i = 0; i < n; ++i)
38                    if (cur[i] & (T(1) << h)) b[h].set_bit(i);
39                b[h].build();
40                int il = 0, ir = b[h].cnt0;
41                for (uint i = 0; i < n; ++i)
42                    nxt[(b[h][i] ? ir : il)++] = cur[i];
43                swap(cur, nxt);
44            }
45        }
46        Expected: min(5, 10, 7, -3, 1) = -3
47        Expected: min(7, 8, 7, 1) = 1
48        Expected: min(5, 10, 7, -3, 1, 0) = 0
49        operator[](uint i) const {
50            T res = 0;
51            for (int h = lg; h--;) {
52                if (b[h][i])
53                    i += b[h].cnt0 - b[h].rank0(i), res |= T(1) << h;
54                else i = b[h].rank0(i);
55            }
56            return res;
57        }
58        // query k-th smallest (0-based) in a[l, r]
59        Expected: kth(0, 10, 5, 0) = 5
60        Expected: kth(0, 10, 5, 1) = 6
61        Expected: kth(0, 10, 5, 2) = 7
62        Expected: kth(0, 10, 5, 3) = 8
63        Expected: kth(0, 10, 5, 4) = 9
64        Expected: kth(0, 10, 5, 5) = 10
65        Expected: kth(0, 10, 5, 6) = 11
66        Expected: kth(0, 10, 5, 7) = 12
67        Expected: kth(0, 10, 5, 8) = 13
68        Expected: kth(0, 10, 5, 9) = 14
69        Expected: kth(0, 10, 5, 10) = 15
70        Expected: kth(0, 10, 5, 11) = 16
71        Expected: kth(0, 10, 5, 12) = 17
72        Expected: kth(0, 10, 5, 13) = 18
73        Expected: kth(0, 10, 5, 14) = 19
74        Expected: kth(0, 10, 5, 15) = 20
75        Expected: kth(0, 10, 5, 16) = 21
76        Expected: kth(0, 10, 5, 17) = 22
77        Expected: kth(0, 10, 5, 18) = 23
78        Expected: kth(0, 10, 5, 19) = 24
79        Expected: kth(0, 10, 5, 20) = 25
80        Expected: kth(0, 10, 5, 21) = 26
81        Expected: kth(0, 10, 5, 22) = 27
82        Expected: kth(0, 10, 5, 23) = 28
83        Expected: kth(0, 10, 5, 24) = 29
84        Expected: kth(0, 10, 5, 25) = 30
85        Expected: kth(0, 10, 5, 26) = 31
86        Expected: kth(0, 10, 5, 27) = 32
87        Expected: kth(0, 10, 5, 28) = 33
88        Expected: kth(0, 10, 5, 29) = 34
89        Expected: kth(0, 10, 5, 30) = 35
90        Expected: kth(0, 10, 5, 31) = 36
91        Expected: kth(0, 10, 5, 32) = 37
92        Expected: kth(0, 10, 5, 33) = 38
93        Expected: kth(0, 10, 5, 34) = 39
94        Expected: kth(0, 10, 5, 35) = 40
95        Expected: kth(0, 10, 5, 36) = 41
96        Expected: kth(0, 10, 5, 37) = 42
97        Expected: kth(0, 10, 5, 38) = 43
98        Expected: kth(0, 10, 5, 39) = 44
99        Expected: kth(0, 10, 5, 40) = 45
100       Expected: kth(0, 10, 5, 41) = 46
101       Expected: kth(0, 10, 5, 42) = 47
102       Expected: kth(0, 10, 5, 43) = 48
103       Expected: kth(0, 10, 5, 44) = 49
104       Expected: kth(0, 10, 5, 45) = 50
105       Expected: kth(0, 10, 5, 46) = 51
106       Expected: kth(0, 10, 5, 47) = 52
107       Expected: kth(0, 10, 5, 48) = 53
108       Expected: kth(0, 10, 5, 49) = 54
109       Expected: kth(0, 10, 5, 50) = 55
110       Expected: kth(0, 10, 5, 51) = 56
111       Expected: kth(0, 10, 5, 52) = 57
112       Expected: kth(0, 10, 5, 53) = 58
113       Expected: kth(0, 10, 5, 54) = 59
114       Expected: kth(0, 10, 5, 55) = 60
115       Expected: kth(0, 10, 5, 56) = 61
116       Expected: kth(0, 10, 5, 57) = 62
117       Expected: kth(0, 10, 5, 58) = 63
118       Expected: kth(0, 10, 5, 59) = 64
119       Expected: kth(0, 10, 5, 60) = 65
120       Expected: kth(0, 10, 5, 61) = 66
121       Expected: kth(0, 10, 5, 62) = 67
122       Expected: kth(0, 10, 5, 63) = 68
123       Expected: kth(0, 10, 5, 64) = 69
124       Expected: kth(0, 10, 5, 65) = 70
125       Expected: kth(0, 10, 5, 66) = 71
126       Expected: kth(0, 10, 5, 67) = 72
127       Expected: kth(0, 10, 5, 68) = 73
128       Expected: kth(0, 10, 5, 69) = 74
129       Expected: kth(0, 10, 5, 70) = 75
130       Expected: kth(0, 10, 5, 71) = 76
131       Expected: kth(0, 10, 5, 72) = 77
132       Expected: kth(0, 10, 5, 73) = 78
133       Expected: kth(0, 10, 5, 74) = 79
134       Expected: kth(0, 10, 5, 75) = 80
135       Expected: kth(0, 10, 5, 76) = 81
136       Expected: kth(0, 10, 5, 77) = 82
137       Expected: kth(0, 10, 5, 78) = 83
138       Expected: kth(0, 10, 5, 79) = 84
139       Expected: kth(0, 10, 5, 80) = 85
140       Expected: kth(0, 10, 5, 81) = 86
141       Expected: kth(0, 10, 5, 82) = 87
142       Expected: kth(0, 10, 5, 83) = 88
143       Expected: kth(0, 10, 5, 84) = 89
144       Expected: kth(0, 10, 5, 85) = 90
145       Expected: kth(0, 10, 5, 86) = 91
146       Expected: kth(0, 10, 5, 87) = 92
147       Expected: kth(0, 10, 5, 88) = 93
148       Expected: kth(0, 10, 5, 89) = 94
149       Expected: kth(0, 10, 5, 90) = 95
150       Expected: kth(0, 10, 5, 91) = 96
151       Expected: kth(0, 10, 5, 92) = 97
152       Expected: kth(0, 10, 5, 93) = 98
153       Expected: kth(0, 10, 5, 94) = 99
154       Expected: kth(0, 10, 5, 95) = 100
155       Expected: kth(0, 10, 5, 96) = 101
156       Expected: kth(0, 10, 5, 97) = 102
157       Expected: kth(0, 10, 5, 98) = 103
158       Expected: kth(0, 10, 5, 99) = 104
159       Expected: kth(0, 10, 5, 100) = 105
160       Expected: kth(0, 10, 5, 101) = 106
161       Expected: kth(0, 10, 5, 102) = 107
162       Expected: kth(0, 10, 5, 103) = 108
163       Expected: kth(0, 10, 5, 104) = 109
164       Expected: kth(0, 10, 5, 105) = 110
165       Expected: kth(0, 10, 5, 106) = 111
166       Expected: kth(0, 10, 5, 107) = 112
167       Expected: kth(0, 10, 5, 108) = 113
168       Expected: kth(0, 10, 5, 109) = 114
169       Expected: kth(0, 10, 5, 110) = 115
170       Expected: kth(0, 10, 5, 111) = 116
171       Expected: kth(0, 10, 5, 112) = 117
172       Expected: kth(0, 10, 5, 113) = 118
173       Expected: kth(0, 10, 5, 114) = 119
174       Expected: kth(0, 10, 5, 115) = 120
175       Expected: kth(0, 10, 5, 116) = 121
176       Expected: kth(0, 10, 5, 117) = 122
177       Expected: kth(0, 10, 5, 118) = 123
178       Expected: kth(0, 10, 5, 119) = 124
179       Expected: kth(0, 10, 5, 120) = 125
180       Expected: kth(0, 10, 5, 121) = 126
181       Expected: kth(0, 10, 5, 122) = 127
182       Expected: kth(0, 10, 5, 123) = 128
183       Expected: kth(0, 10, 5, 124) = 129
184       Expected: kth(0, 10, 5, 125) = 130
185       Expected: kth(0, 10, 5, 126) = 131
186       Expected: kth(0, 10, 5, 127) = 132
187       Expected: kth(0, 10, 5, 128) = 133
188       Expected: kth(0, 10, 5, 129) = 134
189       Expected: kth(0, 10, 5, 130) = 135
190       Expected: kth(0, 10, 5, 131) = 136
191       Expected: kth(0, 10, 5, 132) = 137
192       Expected: kth(0, 10, 5, 133) = 138
193       Expected: kth(0, 10, 5, 134) = 139
194       Expected: kth(0, 10, 5, 135) = 140
195       Expected: kth(0, 10, 5, 136) = 141
196       Expected: kth(0, 10, 5, 137) = 142
197       Expected: kth(0, 10, 5, 138) = 143
198       Expected: kth(0, 10, 5, 139) = 144
199       Expected: kth(0, 10, 5, 140) = 145
200       Expected: kth(0, 10, 5, 141) = 146
201       Expected: kth(0, 10, 5, 142) = 147
202       Expected: kth(0, 10, 5, 143) = 148
203       Expected: kth(0, 10, 5, 144) = 149
204       Expected: kth(0, 10, 5, 145) = 150
205       Expected: kth(0, 10, 5, 146) = 151
206       Expected: kth(0, 10, 5, 147) = 152
207       Expected: kth(0, 10, 5, 148) = 153
208       Expected: kth(0, 10, 5, 149) = 154
209       Expected: kth(0, 10, 5, 150) = 155
210       Expected: kth(0, 10, 5, 151) = 156
211       Expected: kth(0, 10, 5, 152) = 157
212       Expected: kth(0, 10, 5, 153) = 158
213       Expected: kth(0, 10, 5, 154) = 159
214       Expected: kth(0, 10, 5, 155) = 160
215       Expected: kth(0, 10, 5, 156) = 161
216       Expected: kth(0, 10, 5, 157) = 162
217       Expected: kth(0, 10, 5, 158) = 163
218       Expected: kth(0, 10, 5, 159) = 164
219       Expected: kth(0, 10, 5, 160) = 165
220       Expected: kth(0, 10, 5, 161) = 166
221       Expected: kth(0, 10, 5, 162) = 167
222       Expected: kth(0, 10, 5, 163) = 168
223       Expected: kth(0, 10, 5, 164) = 169
224       Expected: kth(0, 10, 5, 165) = 170
225       Expected: kth(0, 10, 5, 166) = 171
226       Expected: kth(0, 10, 5, 167) = 172
227       Expected: kth(0, 10, 5, 168) = 173
228       Expected: kth(0, 10, 5, 169) = 174
229       Expected: kth(0, 10, 5, 170) = 175
230       Expected: kth(0, 10, 5, 171) = 176
231       Expected: kth(0, 10, 5, 172) = 177
232       Expected: kth(0, 10, 5, 173) = 178
233       Expected: kth(0, 10, 5, 174) = 179
234       Expected: kth(0, 10, 5, 175) = 180
235       Expected: kth(0, 10, 5, 176) = 181
236       Expected: kth(0, 10, 5, 177) = 182
237       Expected: kth(0, 10, 5, 178) = 183
238       Expected: kth(0, 10, 5, 179) = 184
239       Expected: kth(0, 10, 5, 180) = 185
240       Expected: kth(0, 10, 5, 181) = 186
241       Expected: kth(0, 10, 5, 182) = 187
242       Expected: kth(0, 10, 5, 183) = 188
243       Expected: kth(0, 10, 5, 184) = 189
244       Expected: kth(0, 10, 5, 185) = 190
245       Expected: kth(0, 10, 5, 186) = 191
246       Expected: kth(0, 10, 5, 187) = 192
247       Expected: kth(0, 10, 5, 188) = 193
248       Expected: kth(0, 10, 5, 189) = 194
249       Expected: kth(0, 10, 5, 190) = 195
250       Expected: kth(0, 10, 5, 191) = 196
251       Expected: kth(0, 10, 5, 192) = 197
252       Expected: kth(0, 10, 5, 193) = 198
253       Expected: kth(0, 10, 5, 194) = 199
254       Expected: kth(0, 10, 5, 195) = 200
255       Expected: kth(0, 10, 5, 196) = 201
256       Expected: kth(0, 10, 5, 197) = 202
257       Expected: kth(0, 10, 5, 198) = 203
258       Expected: kth(0, 10, 5, 199) = 204
259       Expected: kth(0, 10, 5, 200) = 205
260       Expected: kth(0, 10, 5, 201) = 206
261       Expected: kth(0, 10, 5, 202) = 207
262       Expected: kth(0, 10, 5, 203) = 208
263       Expected: kth(0, 10, 5, 204) = 209
264       Expected: kth(0, 10, 5, 205) = 210
265       Expected: kth(0, 10, 5, 206) = 211
266       Expected: kth(0, 10, 5, 207) = 212
267       Expected: kth(0, 10, 5, 208) = 213
268       Expected: kth(0, 10, 5, 209) = 214
269       Expected: kth(0, 10, 5, 210) = 215
270       Expected: kth(0, 10, 5, 211) = 216
271       Expected: kth(0, 10, 5, 212) = 217
272       Expected: kth(0, 10, 5, 213) = 218
273       Expected: kth(0, 10, 5, 214) = 219
274       Expected: kth(0, 10, 5, 215) = 220
275       Expected: kth(0, 10, 5, 216) = 221
276       Expected: kth(0, 10, 5, 217) = 222
277       Expected: kth(0, 10, 5, 218) = 223
278       Expected: kth(0, 10, 5, 219) = 224
279       Expected: kth(0, 10, 5, 220) = 225
280       Expected: kth(0, 10, 5, 221) = 226
281       Expected: kth(0, 10, 5, 222) = 227
282       Expected: kth(0, 10, 5, 223) = 228
283       Expected: kth(0, 10, 5, 224) = 229
284       Expected: kth(0, 10, 5, 225) = 230
285       Expected: kth(0, 10, 5, 226) = 231
286       Expected: kth(0, 10, 5, 227) = 232
287       Expected: kth(0, 10, 5, 228) = 233
288       Expected: kth(0, 10, 5, 229) = 234
289       Expected: kth(0, 10, 5, 230) = 235
290       Expected: kth(0, 10, 5, 231) = 236
291       Expected: kth(0, 10, 5, 232) = 237
292       Expected: kth(0, 10, 5, 233) = 238
293       Expected: kth(0, 10, 5, 234) = 239
294       Expected: kth(0, 10, 5, 235) = 240
295       Expected: kth(0, 10, 5, 236) = 241
296       Expected: kth(0, 10, 5, 237) = 242
297       Expected: kth(0, 10, 5, 238) = 243
298       Expected: kth(0, 10, 5, 239) = 244
299       Expected: kth(0, 10, 5, 240) = 245
300       Expected: kth(0, 10, 5, 241) = 246
301       Expected: kth(0, 10, 5, 242) = 247
302       Expected: kth(0, 10, 5, 243) = 248
303       Expected: kth(0, 10, 5, 244) = 249
304       Expected: kth(0, 10, 5, 245) = 250
305       Expected: kth(0, 10, 5, 246) = 251
306       Expected: kth(0, 10, 5, 247) = 252
307       Expected: kth(0, 10, 5, 248) = 253
308       Expected: kth(0, 10, 5, 249) = 254
309       Expected: kth(0, 10, 5, 250) = 255
310       Expected: kth(0, 10, 5, 251) = 256
311       Expected: kth(0, 10, 5, 252) = 257
312       Expected: kth(0, 10, 5, 253) = 258
313       Expected: kth(0, 10, 5, 254) = 259
314       Expected: kth(0, 10, 5, 255) = 260
315       Expected: kth(0, 10, 5, 256) = 261
316       Expected: kth(0, 10, 5, 257) = 262
317       Expected: kth(0, 10, 5, 258) = 263
318       Expected: kth(0, 10, 5, 259) = 264
319       Expected: kth(0, 10, 5, 260) = 265
320       Expected: kth(0, 10, 5, 261) = 266
321       Expected: kth(0, 10, 5, 262) = 267
322       Expected: kth(0, 10, 5, 263) = 268
323       Expected: kth(0, 10, 5, 264) = 269
324       Expected: kth(0, 10, 5, 265) = 270
325       Expected: kth(0, 10, 5, 266) = 271
326       Expected: kth(0, 10, 5, 267) = 272
327       Expected: kth(0, 10, 5, 268) = 273
328       Expected: kth(0, 10, 5, 269) = 274
329       Expected: kth(0, 10, 5, 270) = 275
330       Expected: kth(0, 10, 5, 271) = 276
331       Expected: kth(0, 10, 5, 272) = 277
332       Expected: kth(0, 10, 5, 273) = 278
333       Expected: kth(0, 10, 5, 274) = 279
334       Expected: kth(0, 10, 5, 275) = 280
335       Expected: kth(0, 10, 5, 276) = 281
336       Expected: kth(0, 10, 5, 277) = 282
337       Expected: kth(0, 10, 5, 278) = 283
338       Expected: kth(0, 10, 5, 279) = 284
339       Expected: kth(0, 10, 5, 280) = 285
340       Expected: kth(0, 10, 5, 281) = 286
341       Expected: kth(0, 10, 5, 282) = 287
342       Expected: kth(0, 10, 5, 283) = 288
343       Expected: kth(0, 10, 5, 284) = 289
344       Expected: kth(0, 10, 5, 285) = 290
345       Expected: kth(0, 10, 5, 286) = 291
346       Expected: kth(0, 10, 5, 287) = 292
347       Expected: kth(0, 10, 5, 288) = 293
348       Expected: kth(0, 10, 5, 289) = 294
349       Expected: kth(0, 10, 5, 290) = 295
350       Expected: kth(0, 10, 5, 291) = 296
351       Expected: kth(0, 10, 5, 292) = 297
352       Expected: kth(0, 10, 5, 293) = 298
353       Expected: kth(0, 10, 5, 294) = 299
354       Expected: kth(0, 10, 5, 295) = 300
355       Expected: kth(0, 10, 5, 296) = 301
356       Expected: kth(0, 10, 5, 297) = 302
357       Expected: kth(0, 10, 5, 298) = 303
358       Expected: kth(0, 10, 5, 299) = 304
359       Expected: kth(0, 10, 5, 300) = 305
360       Expected: kth(0, 10, 5, 301) = 306
361       Expected: kth(0, 10, 5, 302) = 307
362       Expected: kth(0, 10, 5, 303) = 308
363       Expected: kth(0, 10, 5, 304) = 309
364       Expected: kth(0, 10, 5, 305) = 310
365       Expected: kth(0, 10, 5, 306) = 311
366       Expected: kth(0, 10, 5, 307) = 312
367       Expected: kth(0, 10, 5, 308) = 313
368       Expected: kth(0, 10, 5, 309) = 314
369       Expected: kth(0, 10, 5, 310) = 315
370       Expected: kth(0, 10, 5, 311) = 316
371       Expected: kth(0, 10, 5, 312) = 317
372       Expected: kth(0, 10, 5, 313) = 318
373       Expected: kth(0, 10, 5, 314) = 319
374       Expected: kth(0, 10, 5, 315) = 320
375       Expected: kth(0, 10
```

```

5  struct Splay {
7    static Splay nil, mem[MEM], *pmem;
8    Splay *ch[2], *f;
9    int val, rev, size;
10   Splay() : val(-1), rev(0), size(0) {
11     f = ch[0] = ch[1] = &nil;
12   }
13   Splay(int _val) : val(_val), rev(0), size(1) {
14     f = ch[0] = ch[1] = &nil;
15   }
16   bool isr() {
17     return f->ch[0] != this && f->ch[1] != this;
18   }
19   int dir() { return f->ch[0] == this ? 0 : 1; }
20   void setCh(Splay *c, int d) {
21     ch[d] = c;
22     if (c != &nil) c->f = this;
23     pull();
24   }
25   void push() {
26     if (rev) {
27       swap(ch[0], ch[1]);
28       if (ch[0] != &nil) ch[0]->rev ^= 1;
29       if (ch[1] != &nil) ch[1]->rev ^= 1;
30       rev = 0;
31     }
32   }
33   void pull() {
34     size = ch[0]->size + ch[1]->size + 1;
35     if (ch[0] != &nil) ch[0]->f = this;
36     if (ch[1] != &nil) ch[1]->f = this;
37   }
38 } Splay::nil, Splay::mem[MEM], *Splay::pmem = Splay::mem;
39 Splay *nil = &Splay::nil;

40 void rotate(Splay *x) {
41   Splay *p = x->f;
42   int d = x->dir();
43   if (!p->isr()) p->f->setCh(x, p->dir());
44   else x->f = p->f;
45   p->setCh(x->ch[!d], d);
46   x->setCh(p, !d);
47   p->pull();
48   x->pull();
49 }

50 vector<Splay *> splayVec;
51 void splay(Splay *x) {
52   splayVec.clear();
53   for (Splay *q = x;; q = q->f) {
54     splayVec.push_back(q);
55     if (q->isr()) break;
56   }
57   reverse(begin(splayVec), end(splayVec));
58   for (auto it : splayVec) it->push();
59   while (!x->isr()) {
60     if (x->f->isr()) rotate(x);
61     else if (x->dir() == x->f->dir())
62       rotate(x->f), rotate(x);
63     else rotate(x), rotate(x);
64   }
65 }

66 Splay *access(Splay *x) {
67   Splay *q = nil;
68   for (; x != nil; x = x->f) {
69     splay(x);
70     x->setCh(q, 1);
71     q = x;
72   }
73   return q;
74 }

75 void evert(Splay *x) {
76   access(x);
77   splay(x);
78   x->rev ^= 1;
79   x->push();
80   x->pull();
81 }

82 void link(Splay *x, Splay *y) {
83   // evert(x);
84   access(x);
85   splay(x);
86   evert(y);
87   x->setCh(y, 1);
88 }

89 void cut(Splay *x, Splay *y) {
90   // evert(x);
91   access(y);
92   splay(y);
93   y->push();
94   y->ch[0] = y->ch[0]->f = nil;
95 }

96

```

```

99
100 int N, Q;
101 Splay *vt[MXN];
102
103 int ask(Splay *x, Splay *y) {
104   access(x);
105   access(y);
106   splay(x);
107   int res = x->f->val;
108   if (res == -1) res = x->val;
109   return res;
110 }

111 int main(int argc, char **argv) {
112   scanf("%d%d", &N, &Q);
113   for (int i = 1; i <= N; i++)
114     vt[i] = new (Splay::pmem++) Splay(i);
115   while (Q--) {
116     char cmd[105];
117     int u, v;
118     scanf("%s", cmd);
119     if (cmd[1] == 'i') {
120       scanf("%d%d", &u, &v);
121       link(vt[v], vt[u]);
122     } else if (cmd[0] == 'c') {
123       scanf("%d", &v);
124       cut(vt[1], vt[v]);
125     } else {
126       scanf("%d%d", &u, &v);
127       int res = ask(vt[u], vt[v]);
128       printf("%d\n", res);
129     }
130   }
131 }


```

3. Graph

3.1. Modeling

- Maximum/Minimum flow with lower bound / Circulation problem
 1. Construct super source S and sink T .
 2. For each edge (x, y, l, u) , connect $x \rightarrow y$ with capacity $u - l$.
 3. For each vertex v , denote by $in(v)$ the difference between the sum of incoming lower bounds and the sum of outgoing lower bounds.
 4. If $in(v) > 0$, connect $S \rightarrow v$ with capacity $in(v)$, otherwise, connect $v \rightarrow T$ with capacity $-in(v)$.
 - To maximize, connect $t \rightarrow s$ with capacity ∞ (skip this in circulation problem), and let f be the maximum flow from S to T . If $f \neq \sum_{v \in V, in(v) > 0} in(v)$, there's no solution. Otherwise, the maximum flow from s to t is the answer.
 - To minimize, let f be the maximum flow from S to T . Connect $t \rightarrow s$ with capacity ∞ and let the flow from S to T be f' . If $f + f' \neq \sum_{v \in V, in(v) > 0} in(v)$, there's no solution. Otherwise, f' is the answer.
 5. The solution of each edge e is $l_e + f_e$, where f_e corresponds to the flow of edge e on the graph.
- Construct minimum vertex cover from maximum matching M on bipartite graph (X, Y)
 1. Redirect every edge: $y \rightarrow x$ if $(x, y) \in M$, $x \rightarrow y$ otherwise.
 2. DFS from unmatched vertices in X .
 3. $x \in X$ is chosen iff x is unvisited.
 4. $y \in Y$ is chosen iff y is visited.
- Minimum cost cyclic flow
 1. Construct super source S and sink T
 2. For each edge (x, y, c) , connect $x \rightarrow y$ with $(cost, cap) = (c, 1)$ if $c > 0$, otherwise connect $y \rightarrow x$ with $(cost, cap) = (-c, 1)$
 3. For each edge with $c < 0$, sum these cost as K , then increase $d(y)$ by 1, decrease $d(x)$ by 1
 4. For each vertex v with $d(v) > 0$, connect $S \rightarrow v$ with $(cost, cap) = (0, d(v))$
 5. For each vertex v with $d(v) < 0$, connect $v \rightarrow T$ with $(cost, cap) = (0, -d(v))$
 6. Flow from S to T , the answer is the cost of the flow $C + K$
- Maximum density induced subgraph
 1. Binary search on answer, suppose we're checking answer T
 2. Construct a max flow model, let K be the sum of all weights
 3. Connect source $s \rightarrow v$, $v \in G$ with capacity K
 4. For each edge (u, v, w) in G , connect $u \rightarrow v$ and $v \rightarrow u$ with capacity w
 5. For $v \in G$, connect it with sink $v \rightarrow t$ with capacity $K + 2T - (\sum_{e \in E(v)} w(e)) - 2w(v)$
 6. T is a valid answer if the maximum flow $f < K|V|$
- Minimum weight edge cover
 1. For each $v \in V$ create a copy v' , and connect $u' \rightarrow v'$ with weight $w(u, v)$.
 2. Connect $v \rightarrow v'$ with weight $2\mu(v)$, where $\mu(v)$ is the cost of the cheapest edge incident to v .
 3. Find the minimum weight perfect matching on G' .
- Project selection problem
 1. If $p_v > 0$, create edge (s, v) with capacity p_v ; otherwise, create edge (v, t) with capacity $-p_v$.

2. Create edge (u, v) with capacity w with w being the cost of choosing u without choosing v .
 3. The mincut is equivalent to the maximum profit of a subset of projects.
 • 0/1 quadratic programming

$$\sum_x c_x x + \sum_y c_y \bar{y} + \sum_{xy} c_{xy} x \bar{y} + \sum_{xyx'y'} c_{xyx'y'} (x \bar{y} + x' \bar{y'})$$

can be minimized by the mincut of the following graph:

1. Create edge (x, t) with capacity c_x and create edge (s, y) with capacity c_y .
2. Create edge (x, y) with capacity c_{xy} .
3. Create edge (x, y) and edge (x', y') with capacity $c_{xyx'y'}$.

3.2. Matching/Flows

3.2.1. Dinic's Algorithm

```

1 struct Dinic {
2     struct edge {
3         int to, cap, flow, rev;
4     };
5     static constexpr int MAXN = 1000, MAXF = 1e9;
6     vector<edge> v[MAXN];
7     int top[MAXN], deep[MAXN], side[MAXN], s, t;
8     void make_edge(int s, int t, int cap) {
9         v[s].push_back({t, cap, 0, (int)v[t].size()});
10        v[t].push_back({s, 0, 0, (int)v[s].size() - 1});
11    }
12    int dfs(int a, int flow) {
13        if (a == t || !flow) return flow;
14        for (int &i = top[a]; i < v[a].size(); i++) {
15            edge &e = v[a][i];
16            if (deep[a] + 1 == deep[e.to] && e.cap - e.flow) {
17                int x = dfs(e.to, min(e.cap - e.flow, flow));
18                if (x) {
19                    e.flow += x, v[e.to][e.rev].flow -= x;
20                    return x;
21                }
22            }
23        }
24        deep[a] = -1;
25        return 0;
26    }
27    bool bfs() {
28        queue<int> q;
29        fill_n(deep, MAXN, 0);
30        q.push(s), deep[s] = 1;
31        int tmp;
32        while (!q.empty()) {
33            tmp = q.front(), q.pop();
34            for (edge e : v[tmp])
35                if (!deep[e.to] && e.cap != e.flow)
36                    deep[e.to] = deep[tmp] + 1, q.push(e.to);
37        }
38        return deep[t];
39    }
40    int max_flow(int _s, int _t) {
41        s = _s, t = _t;
42        int flow = 0, tflow;
43        while (bfs())
44            fill_n(top, MAXN, 0);
45        while ((tflow = dfs(s, MAXF))) flow += tflow;
46    }
47    void reset() {
48        fill_n(side, MAXN, 0);
49        for (auto &i : v) i.clear();
50    }
51 };
52

```

3.2.2. Minimum Cost Flow

```

1 struct MCF {
2     struct edge {
3         ll to, from, cap, flow, cost, rev;
4     } *fromE[MAXN];
5     vector<edge> v[MAXN];
6     ll n, s, t, flows[MAXN], dis[MAXN], pi[MAXN], flowlim;
7     void make_edge(int s, int t, ll cap, ll cost) {
8         if (!cap) return;
9         v[s].pb(edge{t, s, cap, 0LL, cost, v[t].size()});
10        v[t].pb(edge{s, t, 0LL, 0LL, -cost, v[s].size() - 1});
11    }
12    bitset<MAXN> vis;
13    void dijkstra() {
14        vis.reset();
15        __gnu_pbds::priority_queue<pair<ll, int>> q;
16        vector<decltype(q)::point_iterator> its(n);
17        q.push({0LL, s});

```

```

19        int now = q.top().second;
20        q.pop();
21        if (vis[now]) continue;
22        vis[now] = 1;
23        ll ndis = dis[now] + pi[now];
24        for (edge &e : v[now]) {
25            if (e.flow == e.cap || vis[e.to]) continue;
26            if (dis[e.to] > ndis + e.cost - pi[e.to]) {
27                dis[e.to] = ndis + e.cost - pi[e.to];
28                flows[e.to] = min(flows[now], e.cap - e.flow);
29                fromE[e.to] = &e;
30                if (its[e.to] == q.end())
31                    its[e.to] = q.push({-dis[e.to], e.to});
32                else q.modify(its[e.to], {-dis[e.to], e.to});
33            }
34        }
35    }
36    bool AP(ll &flow) {
37        fill_n(dis, n, INF);
38        fromE[s] = 0;
39        dis[s] = 0;
40        flows[s] = flowlim - flow;
41        dijkstra();
42        if (dis[t] == INF) return false;
43        flow += flows[t];
44        for (edge *e = fromE[t]; e; e = fromE[e->from]) {
45            e->flow += flows[t];
46            v[e->to][e->rev].flow -= flows[t];
47        }
48        for (int i = 0; i < n; i++)
49            pi[i] = min(pi[i] + dis[i], INF);
50        return true;
51    }
52    pll solve(int _s, int _t, ll _flowlim = INF) {
53        s = _s, t = _t, flowlim = _flowlim;
54        pll re;
55        while (re.F != flowlim && AP(re.F));
56        for (int i = 0; i < n; i++)
57            for (edge &e : v[i])
58                if (e.flow != 0) re.S += e.flow * e.cost;
59        re.S /= 2;
60        return re;
61    }
62    void init(int _n) {
63        n = _n;
64        fill_n(pi, n, 0);
65        for (int i = 0; i < n; i++) v[i].clear();
66    }
67    void setpi(int s) {
68        fill_n(pi, n, INF);
69        pi[s] = 0;
70        for (ll it = 0, flag = 1, tdis; flag && it < n; it++) {
71            flag = 0;
72            for (int i = 0; i < n; i++)
73                if (pi[i] != INF)
74                    for (edge &e : v[i])
75                        if (e.cap && (tdis = pi[i] + e.cost) < pi[e.to])
76                            pi[e.to] = tdis, flag = 1;
77        }
78    }
79}

```

3.2.3. Gomory-Hu Tree

Requires: Dinic's Algorithm

```

1
3 int e[MAXN][MAXN];
4 int p[MAXN];
5 Dinic D; // original graph
6 void gomory_hu() {
7     fill(p, p + n, 0);
8     fill(e[0], e[n], INF);
9     for (int s = 1; s < n; s++) {
10        int t = p[s];
11        Dinic F = D;
12        int tmp = F.max_flow(s, t);
13        for (int i = 1; i < s; i++)
14            e[s][i] = e[i][s] = min(tmp, e[t][i]);
15        for (int i = s + 1; i <= n; i++)
16            if (p[i] == t && F.side[i]) p[i] = s;
17    }
18}

```

3.2.4. Global Minimum Cut

```

1
3 // weights is an adjacency matrix, undirected
4 pair<int, vi> getMinCut(vector<vi> &weights) {
5     int N = sz(weights);
6     vi used(N, cut, best_cut);

```

```

7   int best_weight = -1;
9
10  for (int phase = N - 1; phase >= 0; phase--) {
11    vi w = weights[0], added = used;
12    int prev, k = 0;
13    rep(i, 0, phase) {
14      prev = k;
15      k = -1;
16      rep(j, 1, N) if (!added[j] &&
17                     (k == -1 || w[j] > w[k])) k = j;
18      if (i == phase - 1) {
19        rep(j, 0, N) weights[prev][j] += weights[k][j];
20        rep(j, 0, N) weights[j][prev] = weights[prev][j];
21        used[k] = true;
22        cut.push_back(k);
23        if (best_weight == -1 || w[k] < best_weight) {
24          best_cut = cut;
25          best_weight = w[k];
26        }
27      } else {
28        rep(j, 0, N) w[j] += weights[k][j];
29        added[k] = true;
30      }
31    }
32  }
33  return {best_weight, best_cut};

```

3.2.5. Bipartite Minimum Cover

Requires: Dinic's Algorithm

```

1
3 // maximum independent set = all vertices not covered
4 // x : [0, n), y : [0, m]
5 struct Bipartite_vertex_cover {
6     Dinic D;
7     int n, m, s, t, x[maxn], y[maxn];
8     void make_edge(int x, int y) { D.make_edge(x, y + n, 1); }
9     int matching() {
10         int re = D.max_flow(s, t);
11         for (int i = 0; i < n; i++) {
12             for (Dinic::edge &e : D.v[i])
13                 if (e.to != s && e.flow == 1) {
14                     x[i] = e.to - n, y[e.to - n] = i;
15                     break;
16                 }
17         }
18         return re;
19     }
20     // init() and matching() before use
21     void solve(vector<int> &vx, vector<int> &vy) {
22         bitset<maxn * 2 + 10> vis;
23         queue<int> q;
24         for (int i = 0; i < n; i++)
25             if (x[i] == -1) q.push(i), vis[i] = 1;
26         while (!q.empty()) {
27             int now = q.front();
28             q.pop();
29             if (now < n) {
30                 for (Dinic::edge &e : D.v[now])
31                     if (e.to != s && e.to - n != x[now] && !vis[e.to])
32                         vis[e.to] = 1, q.push(e.to);
33             } else {
34                 if (!vis[y[now - n]])
35                     vis[y[now - n]] = 1, q.push(y[now - n]);
36             }
37         }
38         for (int i = 0; i < n; i++)
39             if (!vis[i]) vx.pb(i);
40         for (int i = 0; i < m; i++)
41             if (vis[i + n]) vy.pb(i);
42     }
43     void init(int _n, int _m) {
44         n = _n, m = _m, s = n + m, t = s + 1;
45         for (int i = 0; i < n; i++)
46             x[i] = -1, D.make_edge(s, i, 1);
47         for (int i = 0; i < m; i++)
48             y[i] = -1, D.make_edge(i + n, t, 1);
49     }
50 }
```

3.2.6. Edmonds' Algorithm

```
1
3 struct Edmonds {
4     int n, T;
5     vector<vector<int>> g;
6     vector<int> pa, p, used, base;
7     Edmonds(int n)
8         : n(n), T(0), g(n), pa(n, -1), p(n), used(n),
9           base(n) {}
10    void add(int a, int b) {
```

```

11     g[a].push_back(b);
12     g[b].push_back(a);
13 }
14 int getBase(int i) {
15     while (i != base[i])
16         base[i] = base[base[i]], i = base[i];
17     return i;
18 }
19 vector<int> toJoin;
20 void mark_path(int v, int x, int b, vector<int> &path) {
21     for (; getBase(v) != b; v = p[x]) {
22         p[v] = x, x = pa[v];
23         toJoin.push_back(v);
24         toJoin.push_back(x);
25         if (!used[x]) used[x] = ++T, path.push_back(x);
26     }
27 }
28 bool go(int v) {
29     for (int x : g[v]) {
30         int b, bv = getBase(v), bx = getBase(x);
31         if (bv == bx) {
32             continue;
33         } else if (used[x]) {
34             vector<int> path;
35             toJoin.clear();
36             if (used[bx] < used[bv])
37                 mark_path(v, x, b = bx, path);
38             else mark_path(x, v, b = bv, path);
39             for (int z : toJoin) base[getBase(z)] = b;
40             for (int z : path)
41                 if (go(z)) return 1;
42         } else if (p[x] == -1) {
43             p[x] = v;
44             if (pa[x] == -1) {
45                 for (int y; x != -1; x = v)
46                     y = p[x], v = pa[y], pa[x] = y, pa[y] = x;
47                 return 1;
48             }
49             if (!used[pa[x]]) {
50                 used[pa[x]] = ++T;
51                 if (go(pa[x])) return 1;
52             }
53         }
54     }
55     return 0;
56 }
57 void init_dfs() {
58     for (int i = 0; i < n; i++)
59         used[i] = 0, p[i] = -1, base[i] = i;
60 }
61 bool dfs(int root) {
62     used[root] = ++T;
63     return go(root);
64 }
65 void match() {
66     int ans = 0;
67     for (int v = 0; v < n; v++)
68         for (int x : g[v])
69             if (pa[v] == -1 && pa[x] == -1) {
70                 pa[v] = x, pa[x] = v, ans++;
71                 break;
72             }
73     init_dfs();
74     for (int i = 0; i < n; i++)
75         if (pa[i] == -1 && dfs(i)) ans++, init_dfs();
76     cout << ans * 2 << "\n";
77     for (int i = 0; i < n; i++)
78         if (pa[i] > i)
79             cout << i + 1 << " " << pa[i] + 1 << "\n";
80 }

```

3.2.7. Minimum Weight Matching

```

1 struct Graph {
2     static const int MAXN = 105;
3     int n, e[MAXN][MAXN];
4     int match[MAXN], d[MAXN], onstk[MAXN];
5     vector<int> stk;
6     void init(int _n) {
7         n = _n;
8         for (int i = 0; i < n; i++)
9             for (int j = 0; j < n; j++)
10                 // change to appropriate infinity
11                 // if not complete graph
12                 e[i][j] = 0;
13     }
14     void add_edge(int u, int v, int w) {
15         e[u][v] = e[v][u] = w;
16     }
17     bool SPFA(int u) {
18         if (onstk[u]) return true;
19         stk.push_back(u);
20         onstk[u] = 1;
21         for (int v = 0; v < n; v++)
22             if (d[v] == -1 || d[v] > d[u] + e[u][v]) {
23                 d[v] = d[u] + e[u][v];
24                 if (!onstk[v])
25                     onstk[v] = 1;
26             }
27         onstk[u] = 0;
28     }
29 }
```

```

21 } graph;
22
23     for (int v = 0; v < n; v++) {
24         if (u != v && match[u] != v && !onstk[v]) {
25             int m = match[v];
26             if (d[m] > d[u] - e[v][m] + e[u][v]) {
27                 d[m] = d[u] - e[v][m] + e[u][v];
28                 onstk[v] = 1;
29                 stk.push_back(v);
30                 if (SPFA(m)) return true;
31                 stk.pop_back();
32                 onstk[v] = 0;
33             }
34         }
35         onstk[u] = 0;
36         stk.pop_back();
37         return false;
38     }
39     int solve() {
40         for (int i = 0; i < n; i += 2) {
41             match[i] = i + 1;
42             match[i + 1] = i;
43         }
44         while (true) {
45             int found = 0;
46             for (int i = 0; i < n; i++) onstk[i] = d[i] = 0;
47             for (int i = 0; i < n; i++) {
48                 stk.clear();
49                 if (!onstk[i] && SPFA(i)) {
50                     found = 1;
51                     while (stk.size() >= 2) {
52                         int u = stk.back();
53                         stk.pop_back();
54                         int v = stk.back();
55                         stk.pop_back();
56                         match[u] = v;
57                         match[v] = u;
58                     }
59                 }
60                 if (!found) break;
61             }
62             int ret = 0;
63             for (int i = 0; i < n; i++) ret += e[i][match[i]];
64             ret /= 2;
65             return ret;
66         }
67     }

```

```

45     if (order[girl_id][boy_id] <
46         order[girl_id][girl_current[girl_id]]) {
47         if (girl_current[girl_id] < n)
48             que.push(girl_current[girl_id]);
49         girl_current[girl_id] = boy_id;
50     } else {
51         que.push(boy_id);
52     }
53 }
54 }

55 }

56 int main() {
57     cin >> n;
58

59     for (int i = 0; i < n; i++) {
60         string p, t;
61         cin >> p;
62         male[p] = i;
63         bname[i] = p;
64         for (int j = 0; j < n; j++) {
65             cin >> t;
66             if (!female.count(t)) {
67                 gname[fit] = t;
68                 female[t] = fit++;
69             }
70             favor[i][j] = female[t];
71         }
72     }

73 }

74 for (int i = 0; i < n; i++) {
75     string p, t;
76     cin >> p;
77     for (int j = 0; j < n; j++) {
78         cin >> t;
79         order[female[p]][male[t]] = j;
80     }
81 }

82 initialize();
83 stable_marriage();

84 for (int i = 0; i < n; i++) {
85     cout << bname[i] << " "
86     << gname[favor[i][current[i] - 1]] << endl;
87 }
88 }
```

3.2.8. Stable Marriage

```
1 // normal stable marriage problem
2 /* input:
3
4 Albert Laura Nancy Marcy
5 Brad Marcy Nancy Laura
6 Chuck Laura Marcy Nancy
7 Laura Chuck Albert Brad
8 Marcy Albert Chuck Brad
9 Nancy Brad Albert Chuck
 */
11
13 using namespace std;
14 const int MAXN = 505;
15
16 int n;
17 int favor[MAXN][MAXN]; // favor[boy_id][rank] = girl_id;
18 int order[MAXN][MAXN]; // order[girl_id][boy_id] = rank;
19 int current[MAXN]; // current[boy_id] = rank;
// boy_id will pursue current[boy_id] girl.
21 int girl_current[MAXN]; // girl[girl_id] = boy_id;
23
24 void initialize() {
25     for (int i = 0; i < n; i++) {
26         current[i] = 0;
27         girl_current[i] = n;
28         order[i][n] = n;
29     }
30 }
31
32 map<string, int> male, female;
33 string bname[MAXN], gname[MAXN];
34 int fit = 0;
35
36 void stable_marriage() {
37
38     queue<int> que;
39     for (int i = 0; i < n; i++) que.push(i);
40     while (!que.empty()) {
41         int boy_id = que.front();
42         que.pop();
43
44         int girl_id = favor[boy_id][current[boy_id]];
45         current[boy_id]++;
46
47         if (girl_current[girl_id] == n) {
48             male[bname[boy_id]] = 1;
49             female[gname[girl_id]] = 1;
50             fit++;
51         } else {
52             que.push(girl_current[girl_id]);
53             girl_current[girl_id] = n;
54         }
55     }
56 }
```

3.2.9. Kuhn-Munkres algorithm

```

45     fill(vx, vx + n, 0);
46     fill(vy, vy + n, 0);
47     if (DFS(i)) break;
48     ll d = INF;
49     for (int j = 0; j < n; j++) {
50         if (!vy[j]) d = min(d, slack[j]);
51     for (int j = 0; j < n; j++) {
52         if (vx[j]) lx[j] -= d;
53         if (vy[j]) ly[j] += d;
54         else slack[j] -= d;
55     }
56 }
57 ll res = 0;
58 for (int i = 0; i < n; i++) {
59     res += edge[match[i]][i];
60 }
61 return res;
62 }
63 } graph;

```

3.3. Shortest Path Faster Algorithm

```

1 struct SPFA {
2     static const int maxn = 1010, INF = 1e9;
3     int dis[maxn];
4     bitset<maxn> inq, inneg;
5     queue<int> q, tq;
6     vector<pii> v[maxn];
7     void make_edge(int s, int t, int w) {
8         v[s].emplace_back(t, w);
9     }
10    void dfs(int a) {
11        inneg[a] = 1;
12        for (pii i : v[a])
13            if (!inneg[i.F]) dfs(i.F);
14    }
15    bool solve(int n, int s) { // true if have neg-cycle
16        for (int i = 0; i <= n; i++) dis[i] = INF;
17        dis[s] = 0, q.push(s);
18        for (int i = 0; i < n; i++) {
19            inq.reset();
20            int now;
21            while (!q.empty()) {
22                now = q.front(), q.pop();
23                for (pii &i : v[now])
24                    if (dis[i.F] > dis[now] + i.S) {
25                        dis[i.F] = dis[now] + i.S;
26                        if (!inq[i.F]) tq.push(i.F), inq[i.F] = 1;
27                    }
28            }
29            q.swap(tq);
30        }
31        bool re = !q.empty();
32        inneg.reset();
33        while (!q.empty())
34            if (!inneg[q.front()]) dfs(q.front());
35            q.pop();
36        }
37        return re;
38    }
39    void reset(int n) {
40        for (int i = 0; i <= n; i++) v[i].clear();
41    }
42 }

```

3.4. Strongly Connected Components

```

1 struct TarjanScc {
2     int n, step;
3     vector<int> time, low, instk, stk;
4     vector<vector<int>> e, scc;
5     TarjanScc(int n_) : n(n_), step(0), time(n), low(n), instk(n), e(n) {}
6     void add_edge(int u, int v) { e[u].push_back(v); }
7     void dfs(int x) {
8         time[x] = low[x] = ++step;
9         stk.push_back(x);
10        instk[x] = 1;
11        for (int y : e[x])
12            if (!time[y])
13                dfs(y);
14            low[x] = min(low[x], low[y]);
15        } else if (instk[y])
16            low[x] = min(low[x], time[y]);
17        }
18        if (time[x] == low[x])
19            scc.emplace_back();
20            for (int y = -1; y != x;)
21                y = stk.back();
22                stk.pop_back();
23                instk[y] = 0;

```

```

25             scc.back().push_back(y);
26         }
27     }
28 }
29 void solve() {
30     for (int i = 0; i < n; i++)
31         if (!time[i]) dfs(i);
32     reverse(scc.begin(), scc.end());
33     // scc in topological order
34 }
35 }

```

3.4.1. 2-Satisfiability

Requires: Strongly Connected Components

```

1
3 // 1 based, vertex in SCC = MAXN * 2
4 // (not i) is i + n
5 struct two_SAT {
6     int n, ans[MAXN];
7     SCC S;
8     void imply(int a, int b) { S.make_edge(a, b); }
9     bool solve(int _n) {
10        n = _n;
11        S.solve(n * 2);
12        for (int i = 1; i <= n; i++) {
13            if (S.scc[i] == S.scc[i + n]) return false;
14            ans[i] = (S.scc[i] < S.scc[i + n]);
15        }
16        return true;
17    }
18    void init(int _n) {
19        n = _n;
20        fill_n(ans, n + 1, 0);
21        S.init(n * 2);
22    }
23 } SAT;

```

3.5. Biconnected Components

3.5.1. Articulation Points

```

1 void dfs(int x, int p) {
2     tin[x] = low[x] = ++t;
3     int ch = 0;
4     for (auto u : g[x])
5         if (u.first != p) {
6             if (!ins[u.second])
7                 st.push(u.second), ins[u.second] = true;
8             if (tin[u.first])
9                 low[x] = min(low[x], tin[u.first]);
10                continue;
11            }
12            ++ch;
13            dfs(u.first, x);
14            low[x] = min(low[x], low[u.first]);
15            if (low[u.first] >= tin[x])
16                cut[x] = true;
17                ++sz;
18                while (true) {
19                    int e = st.top();
20                    st.pop();
21                    bcc[e] = sz;
22                    if (e == u.second) break;
23                }
24            }
25            if (ch == 1 && p == -1) cut[x] = false;
26        }
27 }

```

3.5.2. Bridges

```

1 // if there are multi-edges, then they are not bridges
2 void dfs(int x, int p) {
3     tin[x] = low[x] = ++t;
4     st.push(x);
5     for (auto u : g[x])
6         if (u.first != p) {
7             if (tin[u.first])
8                 low[x] = min(low[x], tin[u.first]);
9                 continue;
10            }
11            dfs(u.first, x);
12            low[x] = min(low[x], low[u.first]);
13            if (low[u.first] == tin[u.first]) br[u.second] = true;
14        }
15        if (tin[x] == low[x]) {
16            ++sz;
17            while (st.size()) {
18                int u = st.top();
19                st.pop();
20                bcc[u] = sz;
21            }
22        }
23 }

```

```

21     if (u == x) break;
23 }
}

```

3.6. Triconnected Components

```

1
3 // requires a union-find data structure
5 struct ThreeEdgeCC {
6     int V, ind;
7     vector<int> id, pre, post, low, deg, path;
8     vector<vector<int>> components;
9     UnionFind uf;
10    template <class Graph>
11    void dfs(const Graph &G, int v, int prev) {
12        pre[v] = ++ind;
13        for (int w : G[v])
14            if (w != v) {
15                if (w == prev) {
16                    prev = -1;
17                    continue;
18                }
19                if (pre[w] != -1) {
20                    if (pre[w] < pre[v]) {
21                        deg[v]++;
22                        low[v] = min(low[v], pre[w]);
23                    } else {
24                        deg[v]--;
25                        int &u = path[v];
26                        for (; u != -1 && pre[u] <= pre[w] &&
27                            pre[w] <= post[u];)
28                            uf.join(v, u);
29                            deg[v] += deg[u];
30                            u = path[u];
31                    }
32                    continue;
33                }
34                dfs(G, w, v);
35                if (path[w] == -1 && deg[w] <= 1) {
36                    deg[v] += deg[w];
37                    low[v] = min(low[v], low[w]);
38                    continue;
39                }
40                if (deg[w] == 0) w = path[w];
41                if (low[v] > low[w]) {
42                    low[v] = min(low[v], low[w]);
43                    swap(w, path[v]);
44                }
45                for (; w != -1; w = path[w]) {
46                    uf.join(v, w);
47                    deg[v] += deg[w];
48                }
49            }
50        post[v] = ind;
51    }
52    template <class Graph>
53    ThreeEdgeCC(const Graph &G)
54        : V(G.size()), ind(-1), id(V, -1), pre(V, -1),
55        post(V), low(V, INT_MAX), deg(V, 0), path(V, -1),
56        uf(V) {
57        for (int v = 0; v < V; v++)
58            if (pre[v] == -1) dfs(G, v, -1);
59        components.reserve(uf.cnt);
60        for (int v = 0; v < V; v++)
61            if (uf.find(v) == v) {
62                id[v] = components.size();
63                components.emplace_back(1, v);
64                components.back().reserve(uf.getSize(v));
65            }
66        for (int v = 0; v < V; v++)
67            if (id[v] == -1)
68                components[id[v]] = id[uf.find(v)].push_back(v);
69    }
70 };
}

```

3.7. Centroid Decomposition

```

1 void get_center(int now) {
2     v[now] = true;
3     vtx.push_back(now);
4     sz[now] = 1;
5     mx[now] = 0;
6     for (int u : G[now])
7         if (!v[u]) {
8             get_center(u);
9             mx[now] = max(mx[now], sz[u]);
10            sz[now] += sz[u];
11        }
}

```

```

13 void get_dis(int now, int d, int len) {
14     dis[d][now] = cnt;
15     v[now] = true;
16     for (auto u : G[now])
17         if (!v[u.first]) { get_dis(u, d, len + u.second); }
18 }
19 void dfs(int now, int fa, int d) {
20     get_center(now);
21     int c = -1;
22     for (int i : vtx) {
23         if (max(mx[i], (int)vtx.size() - sz[i]) <=
24             (int)vtx.size() / 2)
25             c = i;
26         v[i] = false;
27     }
28     get_dis(c, d, 0);
29     for (int i : vtx) v[i] = false;
30     v[c] = true;
31     vtx.clear();
32     dep[c] = d;
33     p[c] = fa;
34     for (auto u : G[c])
35         if (u.first != fa && !v[u.first]) {
36             dfs(u.first, c, d + 1);
37         }
}

```

3.8. Minimum Mean Cycle

```

1
3 // d[i][j] == 0 if {i,j} !in E
4 long long d[1003][1003], dp[1003][1003];
5
6 pair<long long, long long> MMWC() {
7     memset(dp, 0x3f, sizeof(dp));
8     for (int i = 1; i <= n; ++i) dp[0][i] = 0;
9     for (int i = 1; i <= n; ++i) {
10         for (int j = 1; j <= n; ++j) {
11             for (int k = 1; k <= n; ++k) {
12                 dp[i][k] = min(dp[i - 1][j] + d[j][k], dp[i][k]);
13             }
14         }
15     }
16     long long au = 1ll << 31, ad = 1;
17     for (int i = 1; i <= n; ++i) {
18         if (dp[n][i] == 0x3f3f3f3f3f3f3f3f) continue;
19         long long u = 0, d = 1;
20         for (int j = n - 1; j >= 0; --j) {
21             if ((dp[n][i] - dp[j][i]) * d > u * (n - j)) {
22                 u = dp[n][i] - dp[j][i];
23                 d = n - j;
24             }
25         }
26         if (u * ad < au * d) au = u, ad = d;
27     }
28     long long g = __gcd(au, ad);
29     return make_pair(au / g, ad / g);
}

```

3.9. Directed MST

```

1 template <typename T> struct DMST {
2     T g[maxn][maxn], fw[maxn];
3     int n, fr[maxn];
4     bool vis[maxn], inc[maxn];
5     void clear() {
6         for (int i = 0; i < maxn; ++i) {
7             for (int j = 0; j < maxn; ++j) g[i][j] = inf;
8             vis[i] = inc[i] = false;
9         }
10    }
11    void addedge(int u, int v, T w) {
12        g[u][v] = min(g[u][v], w);
13    }
14    T operator()(int root, int _n) {
15        n = _n;
16        if (dfs(root) != n) return -1;
17        T ans = 0;
18        while (true) {
19            for (int i = 1; i <= n; ++i) fw[i] = inf, fr[i] = i;
20            for (int i = 1; i <= n; ++i) {
21                if (!inc[i]) {
22                    for (int j = 1; j <= n; ++j) {
23                        if (!inc[j] && i != j && g[j][i] < fw[i]) {
24                            fw[i] = g[j][i];
25                            fr[i] = j;
26                        }
27                    }
28                }
29                int x = -1;
30                for (int i = 1; i <= n; ++i)
31                    if (i != root && !inc[i]) {
}

```

```

int j = i, c = 0;
while (j != root && fr[j] != i && c <= n)
    ++c, j = fr[j];
if (j == root || c > n) continue;
else {
    x = i;
    break;
}
if (!~x) {
    for (int i = 1; i <= n; ++i)
        if (i != root && !inc[i]) ans += fw[i];
    return ans;
}
int y = x;
for (int i = 1; i <= n; ++i) vis[i] = false;
do {
    ans += fw[y];
    y = fr[y];
    vis[y] = inc[y] = true;
} while (y != x);
inc[x] = false;
for (int k = 1; k <= n; ++k)
    if (vis[k]) {
        for (int j = 1; j <= n; ++j)
            if (!vis[j]) {
                if (g[x][j] > g[k][j]) g[x][j] = g[k][j];
                if (g[j][k] < inf &&
                    g[j][k] - fw[k] < g[j][x])
                    g[j][x] = g[j][k] - fw[k];
            }
    }
}
return ans;
}
int dfs(int now) {
    int r = 1;
    vis[now] = true;
    for (int i = 1; i <= n; ++i)
        if (g[now][i] < inf && !vis[i]) r += dfs(i);
    return r;
}

```

3.10. Maximum Clique

```

// source: KACTL

typedef vector<bitset<200>> vb;
struct Maxclique {
    double limit = 0.025, pk = 0;
    struct Vertex {
        int i, d = 0;
    };
    typedef vector<Vertex> vv;
    vb e;
    vv V;
    vector<vi> C;
    vi qmax, q, S, old;
    void init(vv &r) {
        for (auto &v : r) v.d = 0;
        for (auto &v : r)
            for (auto j : r) v.d += e[v.i][j.i];
        sort(all(r), [&](auto a, auto b) { return a.d > b.d; });
        int mxD = r[0].d;
        rep(i, 0, sz(r)) r[i].d = min(i, mxD) + 1;
    }
    void expand(vv &R, int lev = 1) {
        S[lev] += S[lev - 1] - old[lev];
        old[lev] = S[lev - 1];
        while (sz(R)) {
            if (sz(q) + R.back().d <= sz(qmax)) return;
            q.push_back(R.back().i);
            vv T;
            for (auto v : R)
                if (e[R.back().i][v.i]) T.push_back({v.i});
            if (sz(T)) {
                if (S[lev]++ / ++pk < limit) init(T);
                int j = 0, mxk = 1,
                    mnk = max(sz(qmax) - sz(q) + 1, 1);
                C[1].clear(), C[2].clear();
                for (auto v : T) {
                    int k = 1;
                    auto f = [&](int i) { return e[v.i][i]; };
                    while (any_of(all(C[k]), f)) k++;
                    if (k > mxk) mxk = k, C[mxk + 1].clear();
                    if (k < mnk) T[j++].i = v.i;
                    C[k].push_back(v.i);
                }
                if (j > 0) T[j - 1].d = 0;
                rep(k, mnk, mxk + 1) for (int i : C[k]) T[j].i = i,
                    T[j++].d =
            }
            expand(T, lev + 1);
        }
    }
}
```

```

49 } else if (sz(q) > sz(qmax)) qmax = q;
50     q.pop_back(), R.pop_back();
51 }
52 vi maxClique() {
53     init(V), expand(V);
54     return qmax;
55 }
56 Maxclique(vb conn)
57     : e(conn), C(sz(e) + 1), S(sz(C)), old(S) {
58         rep(i, 0, sz(e)) V.push_back({i});
59     }
60 };

```

3.11. Dominator Tree

```

1 // idom[n] is the unique node that strictly dominates n but
2 // does not strictly dominate any other node that strictly
3 // dominates n. idom[n] = 0 if n is entry or the entry
4 // cannot reach n.
5 struct DominatorTree {
6     static const int MAXN = 200010;
7     int n, s;
8     vector<int> g[MAXN], pred[MAXN];
9     vector<int> cov[MAXN];
10    int dfn[MAXN], nfd[MAXN], ts;
11    int par[MAXN];
12    int sdom[MAXN], idom[MAXN];
13    int mom[MAXN], mn[MAXN];
14
15    inline bool cmp(int u, int v) { return dfn[u] < dfn[v]; }
16
17    int eval(int u) {
18        if (mom[u] == u) return u;
19        int res = eval(mom[u]);
20        if (cmp(sdom[mn[mom[u]]], sdom[mn[u]]))
21            mn[u] = mn[mom[u]];
22        return mom[u] = res;
23    }
24
25    void init(int _n, int _s) {
26        n = _n;
27        s = _s;
28        REP1(i, 1, n) {
29            g[i].clear();
30            pred[i].clear();
31            idom[i] = 0;
32        }
33    }
34    void add_edge(int u, int v) {
35        g[u].push_back(v);
36        pred[v].push_back(u);
37    }
38    void DFS(int u) {
39        ts++;
40        dfn[u] = ts;
41        nfd[ts] = u;
42        for (int v : g[u])
43            if (dfn[v] == 0) {
44                par[v] = u;
45                DFS(v);
46            }
47    }
48    void build() {
49        ts = 0;
50        REP1(i, 1, n) {
51            dfn[i] = nfd[i] = 0;
52            cov[i].clear();
53            mom[i] = mn[i] = sdom[i] = i;
54        }
55        DFS(s);
56        for (int i = ts; i >= 2; i--) {
57            int u = nfd[i];
58            if (u == 0) continue;
59            for (int v : pred[u])
60                if (dfn[v]) {
61                    eval(v);
62                    if (cmp(sdom[mn[v]], sdom[u]))
63                        sdom[u] = sdom[mn[v]];
64                }
65            cov[sdom[u]].push_back(u);
66            mom[u] = par[u];
67            for (int w : cov[par[u]]) {
68                eval(w);
69                if (cmp(sdom[mn[w]], par[u])) idom[w] = mn[w];
70                else idom[w] = par[u];
71            }
72            cov[par[u]].clear();
73        }
74        REP1(i, 2, ts) {
75            int u = nfd[i];
76            if (u == 0) continue;
77            if (idom[u] != sdom[u]) idom[u] = idom[idom[u]];
78        }
79    }
80}
```

```
79 }  
} dom;
```

3.12. Manhattan Distance MST

```
1  
3 // returns [(dist, from, to), ...]  
// then do normal mst afterwards  
5 typedef Point<int> P;  
vector<array<int, 3>> manhattanMST(vector<P> ps) {  
    vi id(sz(ps));  
    iota(all(id), 0);  
    vector<array<int, 3>> edges;  
    rep(k, 0, 4) {  
        sort(all(id), [&](int i, int j) {  
            return (ps[i] - ps[j]).x < (ps[j] - ps[i]).y;  
        });  
        map<int, int> sweep;  
        for (int i : id) {  
            for (auto it = sweep.lower_bound(-ps[i].y);  
                  it != sweep.end(); sweep.erase(it++)) {  
                int j = it->second;  
                P d = ps[i] - ps[j];  
                if (d.y > d.x) break;  
                edges.push_back({d.y + d.x, i, j});  
            }  
            sweep[-ps[i].y] = i;  
        }  
        for (P &p : ps)  
            if (k & 1) p.x = -p.x;  
            else swap(p.x, p.y);  
    }  
    return edges;  
}
```

4. Math

4.1. Number Theory

4.1.1. Mod Struct

A list of safe primes: 26003, 27767, 28319, 28979, 29243, 29759, 30467, 910927547, 919012223, 947326223, 990669467, 1007939579, 1019126699, 929760389146037459, 975500632317046523, 989312547895528379

NTT prime p	$p - 1$	primitive root
65537	$1 \lll 16$	3
998244353	$119 \lll 23$	3
2748779069441	$5 \lll 39$	3
194555039024054273	$27 \lll 56$	5

Requires: Extended GCD

4.1.2. Miller-Rabin

Requires: Mod Struct

```
1  
3 // checks if Mod::MOD is prime  
bool is_prime() {  
    if (MOD < 2 || MOD % 2 == 0) return MOD == 2;  
    Mod A[] = {2, 7, 61}; // for int values (< 2^31)  
    // ll: 2, 325, 9375, 28178, 450775, 9780504, 1795265022  
    int s = __builtin_ctzll(MOD - 1), i;  
    for (Mod a : A) {  
        Mod x = a ^ (MOD >> s);  
        for (i = 0; i < s && (x + 1).v > 2; i++) x *= x;  
        if (i && x != -1) return 0;  
    }  
    return 1;  
}
```

4.1.3. Linear Sieve

```
1 constexpr ll MAXN = 1000000;  
bitset<MAXN> is_prime;  
vector<ll> primes;  
ll mpf[MAXN], phi[MAXN], mu[MAXN];  
5  
void sieve() {  
    is_prime.set();  
    is_prime[1] = 0;  
    mu[1] = phi[1] = 1;  
    for (ll i = 2; i < MAXN; i++) {  
        if (is_prime[i]) {  
            mpf[i] = i;  
            primes.push_back(i);  
            phi[i] = i - 1;  
            mu[i] = -1;  
        }  
        for (ll p : primes) {  
            if (p > mpf[i] || i * p >= MAXN) break;  
            is_prime[i * p] = 0;  
            mpf[i * p] = p;  
            mu[i * p] = -mu[i];  
            if (i % p == 0)  
                phi[i * p] = phi[i] * p, mu[i * p] = 0;  
            else phi[i * p] = phi[i] * (p - 1);  
        }  
    }  
}
```

4.1.4. Get Factors

Requires: Linear Sieve

```
1  
3 template <typename T> struct M {  
    static T MOD; // change to constexpr if already known  
    T v;  
    M(T x = 0) {  
        v = (-MOD <= x && x < MOD) ? x : x % MOD;  
        if (v < 0) v += MOD;  
    }  
    explicit operator T() const { return v; }  
    bool operator==(const M &b) const { return v == b.v; }  
    bool operator!=(const M &b) const { return v != b.v; }  
    M operator-() { return M(-v); }  
    M operator+(M b) { return M(v + b.v); }  
    M operator-(M b) { return M(v - b.v); }  
    M operator*(M b) { return M((__int128)v * b.v % MOD); }  
    M operator/(M b) { return *this * (b ^ (MOD - 2)); }  
    // change above implementation to this if MOD is not prime  
    M inv() {  
        auto [p, _, g] = extgcd(v, MOD);  
        return assert(g == 1), p;  
    }  
    friend M operator^(M a, ll b) {  
        M ans(1);  
        for (; b; b >= 1, a *= a)  
            if (b & 1) ans *= a;  
        return ans;  
    }  
    friend M &operator+=(M &a, M b) { return a = a + b; }  
    friend M &operator-=(M &a, M b) { return a = a - b; }  
    friend M &operator*=(M &a, M b) { return a = a * b; }  
    friend M &operator/=(M &a, M b) { return a = a / b; }  
};  
using Mod = M<int>;  
template <> int Mod::MOD = 1'000'000'007;  
int &MOD = Mod::MOD;
```

```
1  
3 vector<ll> all_factors(ll n) {  
    vector<ll> fac = {1};  
    while (n > 1) {  
        const ll p = mpf[n];  
        vector<ll> cur = {1};  
        while (n % p == 0) {  
            n /= p;  
            cur.push_back(cur.back() * p);  
        }  
        vector<ll> tmp;  
        for (auto x : fac)  
            for (auto y : cur) tmp.push_back(x * y);  
        tmp.swap(fac);  
    }  
    return fac;
```

4.1.5. Binary GCD

```
1 // returns the gcd of non-negative a, b  
ull bin_gcd(ull a, ull b) {  
    if (!a || !b) return a + b;  
    int s = __builtin_ctzll(a | b);  
    a >>= __builtin_ctzll(a);  
    while (b) {  
        if ((b >>= __builtin_ctzll(b)) < a) swap(a, b);  
        b -= a;  
    }  
    return a << s;  
}
```

4.1.6. Extended GCD

```

1 // returns (p, q, g): p * a + q * b == g == gcd(a, b)
2 // g is not guaranteed to be positive when a < 0 or b < 0
3 tuple<ll, ll, ll> extgcd(ll a, ll b) {
4     ll s = 1, t = 0, u = 0, v = 1;
5     while (b) {
6         ll q = a / b;
7         swap(a -= q * b, b);
8         swap(s -= q * t, t);
9         swap(u -= q * v, v);
10    }
11   return {s, u, a};
}

```

4.1.7. Chinese Remainder Theorem

Requires: Extended GCD

```

1 // for 0 <= a < m, 0 <= b < n, returns the smallest x >= 0
2 // such that x % m == a and x % n == b
3 ll crt(ll a, ll m, ll b, ll n) {
4     if (n > m) swap(a, b), swap(m, n);
5     auto [x, y, g] = extgcd(m, n);
6     assert((a - b) % g == 0); // no solution
7     x = ((b - a) / g * x) % (n / g) * m + a;
8     return x < 0 ? x + m / g * n : x;
}

```

4.1.8. Baby-Step Giant-Step

Requires: Mod Struct

```

1 // returns x such that a ^ x = b where x \in [l, r)
2 ll bsgs(Mod a, Mod b, ll l = 0, ll r = MOD - 1) {
3     int m = sqrt(r - l) + 1, i;
4     unordered_map<ll, ll> tb;
5     Mod d = (a ^ l) / b;
6     for (i = 0, d = (a ^ l) / b; i < m; i++, d *= a)
7         if (d == 1) return l + i;
8     else tb[(ll)d] = l + i;
9     Mod c = Mod(1) / (a ^ m);
10    for (i = 0, d = 1; i < m; i++, d *= c)
11        if (auto j = tb.find((ll)d); j != tb.end())
12            return j->second + i * m;
13    return assert(0), -1; // no solution
}

```

4.1.9. Pollard's Rho

```

1 ll f(ll x, ll mod) { return (x * x + 1) % mod; }
2 // n should be composite
3 ll pollard_rho(ll n) {
4     if (!(n & 1)) return 2;
5     while (1) {
6         ll y = 2, x = RNG() % (n - 1) + 1, res = 1;
7         for (int sz = 2; res == 1; sz *= 2) {
8             for (int i = 0; i < sz && res <= 1; i++) {
9                 x = f(x, n);
10                res = __gcd(abs(x - y), n);
11            }
12            y = x;
13        }
14        if (res != 0 && res != n) return res;
15    }
}

```

4.1.10. Tonelli-Shanks Algorithm

Requires: Mod Struct

```

1
2 int legendre(Mod a) {
3     if (a == 0) return 0;
4     return (a ^ ((MOD - 1) / 2)) == 1 ? 1 : -1;
}
5 Mod sqrt(Mod a) {
6     assert(legendre(a) != -1); // no solution
7     ll p = MOD, s = p - 1;
8     if (a == 0) return 0;
9     if (p == 2) return 1;
10    if (p % 4 == 3) return a ^ ((p + 1) / 4);
11    int r, m;
12    for (r = 0; !(s & 1); r++) s >= 1;
13    Mod n = 2;
14    while (legendre(n) != -1) n += 1;
15    Mod x = a ^ ((s + 1) / 2), b = a ^ s, g = n ^ s;
16    while (b != 1) {
17        Mod t = b;
18        for (m = 0; t != 1; m++) t *= t;
19        Mod gs = g ^ (1LL << (r - m - 1));
}

```

```

23     g = gs * gs, x *= gs, b *= g, r = m;
24 }
25 }
26 // to get sqrt(X) modulo p^k, where p is an odd prime:
27 // c = X^2 (mod p), c = X^2 (mod p^k), q = p^(k-1)
28 // X = X^q * c^{(p^k-2q+1)/2} (mod p^k)

```

4.1.11. Chinese Sieve

```

1 const ll N = 1000000;
2 // f, g, h multiplicative, h = f (dirichlet convolution) g
3 ll pre_g(ll n);
4 ll pre_h(ll n);
5 // preprocessed prefix sum of f
6 ll pre_f[N];
7 // prefix sum of multiplicative function f
8 ll solve_f(ll n) {
9     static unordered_map<ll, ll> m;
10    if (n < N) return pre_f[n];
11    if (m.count(n)) return m[n];
12    ll ans = pre_h(n);
13    for (ll l = 2, r; l <= n; l = r + 1) {
14        r = n / (n / l);
15        ans -= (pre_g(r) - pre_g(l - 1)) * djs_f(n / l);
16    }
17    return m[n] = ans;
}

```

4.1.12. Rational Number Binary Search

```

1 struct QQ {
2     ll p, q;
3     QQ go(QQ b, ll d) { return {p + b.p * d, q + b.q * d}; }
4 };
5 bool pred(QQ);
6 // returns smallest p/q in [lo, hi] such that
7 // pred(p/q) is true, and 0 <= p, q <= N
8 QQ frac_bs(ll N) {
9     QQ lo{0, 1}, hi{1, 0};
10    if (pred(lo)) return lo;
11    assert(pred(hi));
12    bool dir = 1, L = 1, H = 1;
13    for (; L || H; dir = !dir) {
14        ll len = 0, step = 1;
15        for (int t = 0; t < 2 && (t ? step /= 2 : step *= 2);)
16            if (QQ mid = hi.go(lo, len + step));
17                mid.p > N || mid.q > N || dir ^ pred(mid))
18                    t++;
19        else len += step;
20        swap(lo, hi = hi.go(lo, len));
21        (dir ? L : H) = !!len;
22    }
23    return dir ? hi : lo;
}

```

4.1.13. Farey Sequence

```

1 // returns (e/f), where (a/b, c/d, e/f) are
2 // three consecutive terms in the order n farey sequence
3 // to start, call next_farey(n, 0, 1, 1, n)
4 pll next_farey(ll n, ll a, ll b, ll c, ll d) {
5     ll p = (n + b) / d;
6     return pll(p * c - a, p * d - b);
7 }

```

4.2. Combinatorics

4.2.1. Matroid Intersection

This template assumes 2 weighted matroids of the same type, and that removing an element is much more expensive than checking if one can be added. **Remember to change the implementation details.**

The ground set is $0, 1, \dots, n - 1$, where element i has weight $w[i]$. For the unweighted version, remove weights and change BF/SPFA to BFS.

```

1 constexpr int N = 100;
2 constexpr int INF = 1e9;
3
4 struct Matroid { // represents an independent set
5     Matroid(bitset<N>); // initialize from an independent set
6     bool can_add(int); // if adding will break independence
7     Matroid remove(int); // removing from the set
8 };
9
10 auto matroid_intersection(int n, const vector<int> &w) {
11     bitset<N> S;
12     for (int sz = 1; sz <= n; sz++) {
13         Matroid M1(S), M2(S);
14
15         vector<vector<pii>> e(n + 2);
16         for (int j = 0; j < n; j++)
}

```

```

17   if (!S[j]) {
18     if (M1.can_add(j)) e[n].emplace_back(j, -w[j]);
19     if (M2.can_add(j)) e[j].emplace_back(n + 1, 0);
20   }
21   for (int i = 0; i < n; i++) {
22     if (S[i]) {
23       Matroid T1 = M1.remove(i), T2 = M2.remove(i);
24       for (int j = 0; j < n; j++) {
25         if (!S[j]) {
26           if (T1.can_add(j)) e[i].emplace_back(j, -w[j]);
27           if (T2.can_add(j)) e[j].emplace_back(i, w[i]);
28         }
29       }
30     }
31     vector<pii> dis(n + 2, {INF, 0});
32     vector<int> prev(n + 2, -1);
33     dis[n] = {0, 0};
34     // change to SPFA for more speed, if necessary
35     bool upd = 1;
36     while (upd) {
37       upd = 0;
38       for (int u = 0; u < n + 2; u++)
39         for (auto [v, c] : e[u]) {
40           pii x(dis[u].first + c, dis[u].second + 1);
41           if (x < dis[v]) dis[v] = x, prev[v] = u, upd = 1;
42         }
43     }
44     if (dis[n + 1].first < INF)
45       for (int x = prev[n + 1]; x != n; x = prev[x])
46         S.flip(x);
47     else break;
48   }
49   // S is the max-weighted independent set with size sz
50   return S;
51 }
52 }
```

4.2.2. De Brujin Sequence

```

1 int res[kN], aux[kN], a[kN], sz;
2 void Rec(int t, int p, int n, int k) {
3   if (t > n) {
4     if (n % p == 0)
5       for (int i = 1; i <= p; ++i) res[sz++] = aux[i];
6     } else {
7       aux[t] = aux[t - p];
8       Rec(t + 1, p, n, k);
9       for (aux[t] = aux[t - p] + 1; aux[t] < k; ++aux[t])
10      Rec(t + 1, t, n, k);
11    }
12  }
13 int DeBruijn(int k, int n) {
14   // return cyclic string of length k^n such that every
15   // string of length n using k character appears as a
16   // substring.
17   if (k == 1) return res[0] = 0, 1;
18   fill(aux, aux + k * n, 0);
19   return sz = 0, Rec(1, 1, n, k), sz;
20 }
```

4.2.3. Multinomial

```

1
2
3 // ways to permute v[i]
4 ll multinomial(vi &v) {
5   ll c = 1, m = v.empty() ? 1 : v[0];
6   for (int i = 1; i < v.size(); i++)
7     for (int j = 0; i < v[i]; j++) c = c * ++m / (j + 1);
8   return c;
9 }
```

4.3. Algebra

4.3.1. Formal Power Series

```

1
2
3 template <typename mint>
4 struct FormalPowerSeries : vector<mint> {
5   using vector<mint>::vector;
6   using FPS = FormalPowerSeries;
7
8   FPS &operator+=(const FPS &r) {
9     if (r.size() > this->size()) this->resize(r.size());
10    for (int i = 0; i < (int)r.size(); i++)
11      (*this)[i] += r[i];
12    return *this;
13  }
14
15   FPS &operator+=(const mint &r) {
```

```

17   if (this->empty()) this->resize(1);
18   (*this)[0] += r;
19   return *this;
20 }
21
22 FPS &operator-=(const FPS &r) {
23   if (r.size() > this->size()) this->resize(r.size());
24   for (int i = 0; i < (int)r.size(); i++)
25     (*this)[i] -= r[i];
26   return *this;
27 }
28
29 FPS &operator-=(const mint &r) {
30   if (this->empty()) this->resize(1);
31   (*this)[0] -= r;
32   return *this;
33 }
34
35 FPS &operator*=(const mint &v) {
36   for (int k = 0; k < (int)this->size(); k++)
37     (*this)[k] *= v;
38   return *this;
39 }
40
41 FPS &operator/=(const FPS &r) {
42   if (this->size() < r.size()) {
43     this->clear();
44     return *this;
45   }
46   int n = this->size() - r.size() + 1;
47   if ((int)r.size() <= 64) {
48     FPS f(*this), g(r);
49     g.shrink();
50     mint coeff = g.back().inverse();
51     for (auto &x : g) x *= coeff;
52     int deg = (int)f.size() - (int)g.size() + 1;
53     int gs = g.size();
54     FPS quo(deg);
55     for (int i = deg - 1; i >= 0; i--) {
56       quo[i] = f[i + gs - 1];
57       for (int j = 0; j < gs; j++)
58         f[i + j] -= quo[i] * g[j];
59     }
60     *this = quo * coeff;
61     this->resize(n, mint(0));
62     return *this;
63   }
64   return *this = ((*this).rev().pre(n) * r.rev().inv(n))
65     .pre(n)
66     .rev();
67 }
68
69 FPS &operator%=(const FPS &r) {
70   *this -= *this / r * r;
71   shrink();
72   return *this;
73 }
74
75 FPS operator+(const FPS &r) const {
76   return FPS(*this) += r;
77 }
78
79 FPS operator-(const mint &v) const {
80   return FPS(*this) -= v;
81 }
82
83 FPS operator-(const FPS &r) const {
84   return FPS(*this) -= r;
85 }
86
87 FPS operator*(const FPS &r) const {
88   return FPS(*this) *= r;
89 }
90
91 FPS operator*(const mint &v) const {
92   return FPS(*this) *= v;
93 }
94
95 FPS operator/(const FPS &r) const {
96   return FPS(*this) /= r;
97 }
98
99 FPS operator%(const FPS &r) const {
100  return FPS(*this) %= r;
101 }
102
103 FPS operator() const {
104   FPS ret(this->size());
105   for (int i = 0; i < (int)this->size(); i++)
106     ret[i] = -(*this)[i];
107   return ret;
108 }
109
110 void shrink() {
111   while (this->size() && this->back() == mint(0))
112     this->pop_back();
113 }
```

```

111 FPS rev() const {
112   FPS ret(*this);
113   reverse(begin(ret), end(ret));
114   return ret;
115 }

117 FPS dot(FPS r) const {
118   FPS ret(min(this->size(), r.size()));
119   for (int i = 0; i < (int)ret.size(); i++)
120     ret[i] = (*this)[i] * r[i];
121   return ret;
122 }

123 FPS pre(int sz) const {
124   return FPS(begin(*this),
125             begin(*this) + min((int)this->size(), sz));
126 }

129 FPS operator>>(int sz) const {
130   if ((int)this->size() <= sz) return {};
131   FPS ret(*this);
132   ret.erase(ret.begin(), ret.begin() + sz);
133   return ret;
134 }

135 FPS operator<<(int sz) const {
136   FPS ret(*this);
137   ret.insert(ret.begin(), sz, mint(0));
138   return ret;
139 }

141 FPS diff() const {
142   const int n = (int)this->size();
143   FPS ret(max(0, n - 1));
144   mint one(1), coeff(1);
145   for (int i = 1; i < n; i++) {
146     ret[i - 1] = (*this)[i] * coeff;
147     coeff += one;
148   }
149   return ret;
150 }

153 FPS integral() const {
154   const int n = (int)this->size();
155   FPS ret(n + 1);
156   ret[0] = mint(0);
157   if (n > 0) ret[1] = mint(1);
158   auto mod = mint::get_mod();
159   for (int i = 2; i <= n; i++)
160     ret[i] = (-ret[mod % i]) * (mod / i);
161   for (int i = 0; i < n; i++) ret[i + 1] *= (*this)[i];
162   return ret;
163 }

165 mint eval(mint x) const {
166   mint r = 0, w = 1;
167   for (auto &v : *this) r += w * v, w *= x;
168   return r;
169 }

171 FPS log(int deg = -1) const {
172   assert((*this)[0] == mint(1));
173   if (deg == -1) deg = (int)this->size();
174   return (this->diff() * this->inv(deg))
175     .pre(deg - 1)
176     .integral();
177 }

179 FPS pow(int64_t k, int deg = -1) const {
180   const int n = (int)this->size();
181   if (deg == -1) deg = n;
182   for (int i = 0; i < n; i++) {
183     if ((*this)[i] != mint(0)) {
184       if (i * k > deg) return FPS(deg, mint(0));
185       mint rev = mint(1) / (*this)[i];
186       FPS ret =
187         (((*this * rev) >> i).log(deg) * k).exp(deg) *
188         ((*this)[i].pow(k));
189       ret = (ret << (i * k)).pre(deg);
190       if ((int)ret.size() < deg) ret.resize(deg, mint(0));
191       return ret;
192     }
193   }
194   return FPS(deg, mint(0));
195 }

197 static void *ntt_ptr;
198 static void set_fft();
199 FPS &operator=(const FPS &r);
200 void ntt();
201 void intt();
202 void ntt_doubling();
203 static int ntt_pr();
204 FPS inv(int deg = -1) const;

```

```

205   FPS exp(int deg = -1) const;
206 }
207 template <typename mint>
208 void *FormalPowerSeries<mint>::ntt_ptr = nullptr;

```

4.4. Theorems

4.4.1. Kirchhoff's Theorem

Denote L be a $n \times n$ matrix as the Laplacian matrix of graph G , where $L_{ii} = d(i)$, $L_{ij} = -c$ where c is the number of edge (i, j) in G .

- The number of undirected spanning in G is $|\det(\tilde{L}_{11})|$.
- The number of directed spanning tree rooted at r in G is $|\det(\tilde{L}_{rr})|$.

4.4.2. Tutte's Matrix

Let D be a $n \times n$ matrix, where $d_{ij} = x_{ij}$ (x_{ij} is chosen uniformly at random) if $i < j$ and $(i, j) \in E$, otherwise $d_{ij} = -d_{ji}$. $\frac{\text{rank}(D)}{2}$ is the maximum matching on G .

4.4.3. Cayley's Formula

- Given a degree sequence d_1, d_2, \dots, d_n for each *labeled* vertices, there are

$$\frac{(n-2)!}{(d_1-1)!(d_2-1)!\cdots(d_n-1)!}$$

spanning trees.

- Let $T_{n,k}$ be the number of *labeled* forests on n vertices with k components, such that vertex $1, 2, \dots, k$ belong to different components. Then $T_{n,k} = kn^{n-k-1}$.

4.4.4. Erdős–Gallai Theorem

A sequence of non-negative integers $d_1 \geq d_2 \geq \dots \geq d_n$ can be represented as the degree sequence of a finite simple graph on n vertices if and only if $d_1 + d_2 + \dots + d_n$ is even and

$$\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k)$$

holds for all $1 \leq k \leq n$.

4.4.5. Burnside's Lemma

Let X be a set and G be a group that acts on X . For $g \in G$, denote by X^g the elements fixed by g :

$$X^g = \{x \in X \mid gx \in X\}$$

Then

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|.$$

5. Numeric

5.1. Barrett Reduction

```

1 using ull = unsigned long long;
2 using ul = __uint128_t;
3 // very fast calculation of a % m
4 struct reduction {
5   const ull m, d;
6   explicit reduction(ull m) : m(m), d(((ul)1 << 64) / m) {}
7   inline ull operator()(ull a) const {
8     ull q = (ull)((ul)d * a) >> 64;
9     return (a - q * m) >= m ? a - m : a;
10 }
11 };

```

5.2. Long Long Multiplication

```

1 using ull = unsigned long long;
2 using ll = long long;
3 using ld = long double;
4 // returns a * b % M where a, b < M < 2**63
5 ull mult(ull a, ull b, ull M) {
6   ll ret = a * b - M * ull(ld(a) * ld(b) / ld(M));
7   return ret + M * (ret < 0) - M * (ret >= (ll)M);
8 }

```

5.3. Fast Fourier Transform

```

1 template <typename T>
2 void fft_(int n, vector<T> &a, vector<T> &rt, bool inv) {
3     vector<int> br(n);
4     for (int i = 1; i < n; i++) {
5         br[i] = (i & 1) ? br[i - 1] + n / 2 : br[i / 2] / 2;
6         if (br[i] > i) swap(a[i], a[br[i]]);
7     }
8     for (int len = 2; len <= n; len *= 2)
9         for (int i = 0; i < n; i += len)
10            for (int j = 0; j < len / 2; j++) {
11                int pos = n / len * (inv ? len - j : j);
12                T u = a[i + j], v = a[i + j + len / 2] * rt[pos];
13                a[i + j] = u + v, a[i + j + len / 2] = u - v;
14            }
15     if (T minv = T(1) / T(n); inv)
16         for (T &x : a) x *= minv;
17 }

```

Requires: Mod Struct

```

1
3 void ntt(vector<Mod> &a, bool inv, Mod primitive_root) {
4     int n = a.size();
5     Mod root = primitive_root ^ (MOD - 1) / n;
6     vector<Mod> rt(n + 1, 1);
7     for (int i = 0; i < n; i++) rt[i + 1] = rt[i] * root;
8     fft_(n, a, rt, inv);
9 }
10 void fft(vector<complex<double>> &a, bool inv) {
11     int n = a.size();
12     vector<complex<double>> rt(n + 1);
13     double arg = acos(-1) * 2 / n;
14     for (int i = 0; i <= n; i++)
15         rt[i] = {cos(arg * i), sin(arg * i)};
16     fft_(n, a, rt, inv);
17 }

```

5.4. Fast Walsh-Hadamard Transform

Requires: Mod Struct

```

1
3 void fwht(vector<Mod> &a, bool inv) {
4     int n = a.size();
5     for (int d = 1; d < n; d <= 1)
6         for (int m = 0; m < n; m++) {
7             if (!(m & d)) {
8                 inv ? a[m] -= a[m | d] : a[m] += a[m | d]; // AND
9                 inv ? a[m | d] -= a[m] : a[m | d] += a[m]; // OR
10                Mod x = a[m], y = a[m | d]; // XOR
11                a[m] = x + y, a[m | d] = x - y; // XOR
12            }
13         if (Mod iv = Mod(1) / n; inv) // XOR
14             for (Mod &i : a) i *= iv; // XOR
15     }

```

5.5. Subset Convolution

Requires: Mod Struct

```

1 #pragma GCC target("popcnt")
2 #include <immintrin.h>
3
5 void fwht(int n, vector<vector<Mod>> &a, bool inv) {
6     for (int h = 0; h < n; h++)
7         for (int i = 0; i < (1 << n); i++)
8             if (!(i & (1 << h)))
9                 for (int k = 0; k <= n; k++)
10                    inv ? a[i | (1 << h)][k] -= a[i][k]
11                      : a[i | (1 << h)][k] += a[i][k];
12
13 // c[k] = sum(popcnt(i & j) == sz && i | j == k) a[i] * b[j]
14 vector<Mod> subset_convolution(int n, int sz,
15                                 const vector<Mod> &a_,
16                                 const vector<Mod> &b_) {
17     int len = n + sz + 1, N = 1 << n;
18     vector<vector<Mod>> a(1 << n, vector<Mod>(len, 0)), b = a;
19     for (int i = 0; i < N; i++)
20         a[i][_mm_popcnt_u64(i)] = a_[i],
21         b[i][_mm_popcnt_u64(i)] = b_[i];
22     fwht(n, a, 0), fwht(n, b, 0);
23     for (int i = 0; i < N; i++) {
24         vector<Mod> tmp(len);
25         for (int j = 0; j < len; j++)
26             for (int k = 0; k <= j; k++)
27                 tmp[j] += a[i][k] * b[i][j - k];
28         a[i] = tmp;
29     }
30     fwht(n, a, 1);
31     vector<Mod> c(N);
32     for (int i = 0; i < N; i++)
33         c[i] = a[i][_mm_popcnt_u64(i) + sz];
34     return c;
35 }

```

5.6. Linear Recurrences

5.6.1. Berlekamp-Massey Algorithm

```

1 template <typename T>
2 vector<T> berlekamp_massey(const vector<T> &s) {
3     int n = s.size(), l = 0, m = 1;
4     vector<T> r(n), p(n);
5     r[0] = p[0] = 1;
6     T b = 1, d = 0;
7     for (int i = 0; i < n; i++, m++, d = 0) {
8         for (int j = 0; j <= l; j++) d += r[j] * s[i - j];
9         if ((d /= b) == 0) continue; // change if T is float
10        auto t = r;
11        for (int j = m; j < n; j++) r[j] -= d * p[j - m];
12        if (l * 2 <= i) l = i + 1 - l, b *= d, m = 0, p = t;
13    }
14    return r.resize(l + 1), reverse(r.begin(), r.end()), r;
15 }

```

5.6.2. Linear Recurrence Calculation

```

1 template <typename T> struct lin_rec {
2     using poly = vector<T>;
3     poly mul(poly a, poly b, poly m) {
4         int n = m.size();
5         poly r(n);
6         for (int i = n - 1; i >= 0; i--) {
7             r.insert(r.begin(), 0), r.pop_back();
8             T c = r[n - 1] + a[n - 1] * b[i];
9             // c /= m[n - 1]; if m is not monic
10            for (int j = 0; j < n; j++)
11                r[j] += a[j] * b[i] - c * m[j];
12        }
13        return r;
14    }
15    poly pow(poly p, ll k, poly m) {
16        poly r(m.size());
17        r[0] = 1;
18        for (; k; k >= 1, p = mul(p, p, m))
19            if (k & 1) r = mul(r, p, m);
20        return r;
21    }
22    T calc(poly t, poly r, ll k) {
23        int n = r.size();
24        poly p(n);
25        p[1] = 1;
26        poly q = pow(p, k, r);
27        T ans = 0;
28        for (int i = 0; i < n; i++) ans += t[i] * q[i];
29        return ans;
30    }
31 }

```

5.7. Matrices

5.7.1. Determinant

Requires: Mod Struct

```

1
3 Mod det(vector<vector<Mod>> a) {
4     int n = a.size();
5     Mod ans = 1;
6     for (int i = 0; i < n; i++) {
7         int b = i;
8         for (int j = i + 1; j < n; j++) {
9             if (a[j][i] != 0) {
10                 b = j;
11                 break;
12             }
13         if (i != b) swap(a[i], a[b]), ans = -ans;
14         ans *= a[i][i];
15         if (ans == 0) return 0;
16         for (int j = i + 1; j < n; j++) {
17             Mod v = a[j][i] / a[i][i];
18             if (v != 0)
19                 for (int k = i + 1; k < n; k++)
20                     a[j][k] -= v * a[i][k];
21         }
22     }
23     return ans;
24 }

```

```

1
3 double det(vector<vector<double>> a) {
4     int n = a.size();
5     double ans = 1;
6     for (int i = 0; i < n; i++) {
7         int b = i;
8         for (int j = i + 1; j < n; j++) {
9             if (fabs(a[j][i]) > fabs(a[b][i])) b = j;
10            if (i != b) swap(a[i], a[b]), ans = -ans;
11            ans *= a[i][i];
12        }
13    }
14    return ans;
15 }

```

```

11 if (ans == 0) return 0;
12 for (int j = i + 1; j < n; j++) {
13     double v = a[j][i] / a[i][i];
14     if (v != 0)
15         for (int k = i + 1; k < n; k++)
16             a[j][k] -= v * a[i][k];
17 }
18 return ans;
19 }
```

5.7.2. Inverse

```

1
3 // Returns rank.
4 // Result is stored in A unless singular (rank < n).
5 // For prime powers, repeatedly set
6 // A^{-1} = A^{-1} (2I - A^A{-1}) (mod p^k)
7 // where A^{-1} starts as the inverse of A mod p,
8 // and k is doubled in each step.
9
10 int matInv(vector<vector<double>> &A) {
11     int n = sz(A);
12     vi col(n);
13     vector<vector<double>> tmp(n, vector<double>(n));
14     rep(i, 0, n) tmp[i][i] = 1, col[i] = i;
15
16     rep(i, 0, n) {
17         int r = i, c = i;
18         rep(j, i, n)
19             rep(k, i, n) if (fabs(A[j][k]) > fabs(A[r][c])) r = j, c = k;
20
21         if (fabs(A[r][c]) < 1e-12) return i;
22         A[i].swap(A[r]);
23         tmp[i].swap(tmp[r]);
24         rep(j, 0, n) swap(A[j][i], A[j][c]),
25             swap(tmp[j][i], tmp[j][c]);
26         swap(col[i], col[c]);
27         double v = A[i][i];
28         rep(j, i + 1, n) {
29             double f = A[j][i] / v;
30             A[j][i] = 0;
31             rep(k, i + 1, n) A[j][k] -= f * A[i][k];
32             rep(k, 0, n) tmp[j][k] -= f * tmp[i][k];
33         }
34         rep(j, i + 1, n) A[i][j] /= v;
35         rep(j, 0, n) tmp[i][j] /= v;
36         A[i][i] = 1;
37     }
38
39     for (int i = n - 1; i > 0; --i) rep(j, 0, i) {
40         double v = A[j][i];
41         rep(k, 0, n) tmp[j][k] -= v * tmp[i][k];
42     }
43
44     rep(i, 0, n) rep(j, 0, n) A[col[i]][col[j]] = tmp[i][j];
45     return n;
46 }
47
48 int matInv_mod(vector<vector<ll>> &A) {
49     int n = sz(A);
50     vi col(n);
51     vector<vector<ll>> tmp(n, vector<ll>(n));
52     rep(i, 0, n) tmp[i][i] = 1, col[i] = i;
53
54     rep(i, 0, n) {
55         int r = i, c = i;
56         rep(j, i, n) rep(k, i, n) if (A[j][k]) {
57             r = j;
58             c = k;
59             goto found;
60         }
61         return i;
62     }
63     found:
64     A[i].swap(A[r]);
65     tmp[i].swap(tmp[r]);
66     rep(j, 0, n) swap(A[j][i], A[j][c]),
67         swap(tmp[j][i], tmp[j][c]);
68     swap(col[i], col[c]);
69     ll v = modpow(A[i][i], mod - 2);
70     rep(j, i + 1, n) {
71         ll f = A[j][i] * v % mod;
72         A[j][i] = 0;
73         rep(k, i + 1, n) A[j][k] =
74             (A[j][k] - f * A[i][k]) % mod;
75         rep(k, 0, n) tmp[j][k] =
76             (tmp[j][k] - f * tmp[i][k]) % mod;
77     }
78     rep(j, i + 1, n) A[i][j] = A[i][j] * v % mod;
79     rep(j, 0, n) tmp[i][j] = tmp[i][j] * v % mod;
80     A[i][i] = 1;
81 }
```

```

83     for (int i = n - 1; i > 0; --i) rep(j, 0, i) {
84         ll v = A[j][i];
85         rep(k, 0, n) tmp[j][k] =
86             (tmp[j][k] - v * tmp[i][k]) % mod;
87     }
88
89     rep(i, 0, n) rep(j, 0, n) A[col[i]][col[j]] =
90         tmp[i][j] % mod + (tmp[i][j] < 0 ? mod : 0);
91     return n;
92 }
```

5.7.3. Characteristic Polynomial

```

1
3
4 // calculate det(a - xI)
5 template <typename T>
6 vector<T> CharacteristicPolynomial(vector<vector<T>> a) {
7     int N = a.size();
8
9     for (int j = 0; j < N - 2; j++) {
10         for (int i = j + 1; i < N; i++) {
11             if (a[i][j] != 0) {
12                 swap(a[j + 1], a[i]);
13                 for (int k = 0; k < N; k++)
14                     swap(a[k][j + 1], a[k][i]);
15                 break;
16             }
17         }
18         if (a[j + 1][j] != 0) {
19             T inv = T(1) / a[j + 1][j];
20             for (int i = j + 2; i < N; i++) {
21                 if (a[i][j] == 0) continue;
22                 T coe = inv * a[i][j];
23                 for (int l = j; l < N; l++)
24                     a[i][l] -= coe * a[j + 1][l];
25                 for (int k = 0; k < N; k++)
26                     a[k][j + 1] += coe * a[k][i];
27             }
28         }
29     }
30
31     vector<vector<T>> p(N + 1);
32     p[0] = {T(1)};
33     for (int i = 1; i <= N; i++) {
34         p[i].resize(i + 1);
35         for (int j = 0; j < i; j++) {
36             p[i][j + 1] -= p[i - 1][j];
37             p[i][j] += p[i - 1][j] * a[i - 1][i - 1];
38         }
39         T x = 1;
40         for (int m = 1; m < i; m++) {
41             x *= -a[i - m][i - m - 1];
42             T coe = x * a[i - m - 1][i - 1];
43             for (int j = 0; j < i - m; j++) {
44                 p[i][j] += coe * p[i - m - 1][j];
45             }
46         }
47     }
48     return p[N];
49 }
```

5.7.4. Solve Linear Equation

```

1
3 typedef vector<double> vd;
4 const double eps = 1e-12;
5
6 // solves for x: A * x = b
7 int solveLinear(vector<vd> &A, vd &b, vd &x) {
8     int n = sz(A), m = sz(x), rank = 0, br, bc;
9     if (n < m) assert(sz(A[0]) == m);
10    vi col(m);
11    iota(all(col), 0);
12
13    rep(i, 0, n) {
14        double v, bv = 0;
15        rep(r, i, n) rep(c, i, m) if ((v = fabs(A[r][c])) > bv)
16            br = r,
17            bc = c, bv = v;
18        if (bv <= eps) {
19            rep(j, i, n) if (fabs(b[j]) > eps) return -1;
20            break;
21        }
22        swap(A[i], A[br]);
23        swap(b[i], b[br]);
24        swap(col[i], col[bc]);
25        rep(j, 0, n) swap(A[j][i], A[j][bc]);
26        bv = 1 / A[i][i];
27        rep(j, i + 1, n) {
28            double fac = A[j][i] * bv;
```

```

29     b[j] -= fac * b[i];
31     rep(k, i + 1, m) A[j][k] -= fac * A[i][k];
33   }
35   x.assign(m, 0);
36   for (int i = rank; i--;) {
37     b[i] /= A[i][i];
38     x[col[i]] = b[i];
39     rep(j, 0, i) b[j] -= A[j][i] * b[i];
40   }
41   return rank; // (multiple solutions if rank < m)
}

```

5.8. Polynomial Interpolation

```

1
3 // returns a, such that a[0]x^0 + a[1]x^1 + a[2]x^2 + ...
// passes through the given points
5 typedef vector<double> vd;
7 vd interpolate(vd x, vd y, int n) {
8   vd res(n), temp(n);
9   rep(k, 0, n - 1) rep(i, k + 1, n) y[i] =
10    (y[i] - y[k]) / (x[i] - x[k]);
11   double last = 0;
12   temp[0] = 1;
13   rep(k, 0, n) rep(i, 0, n) {
14     res[i] += y[k] * temp[i];
15     swap(last, temp[i]);
16     temp[i] -= last * x[k];
17   }
18   return res;
}

```

5.9. Simplex Algorithm

```

1 // Two-phase simplex algorithm for solving linear programs
// of the form
3 //
5 // maximize      c^T x
6 // subject to    Ax <= b
7 //                  x >= 0
8 //
9 // INPUT: A -- an m x n matrix
10 //        b -- an m-dimensional vector
11 //        c -- an n-dimensional vector
12 //        x -- a vector where the optimal solution will be
13 //              stored
14 //
15 // OUTPUT: value of the optimal solution (infinity if
16 // unbounded
17 //          above, nan if infeasible)
18 //
19 // To use this code, create an LPSolver object with A, b,
20 // and c as arguments. Then, call Solve(x).
21
22 typedef long double ld;
23 typedef vector<ld> vd;
24 typedef vector<vd> vvd;
25 typedef vector<int> vi;
26
27 const ld EPS = 1e-9;
28
29 struct LPSolver {
30   int m, n;
31   vi B, N;
32   vvd D;
33
34   LPSolver(const vvd &A, const vd &b, const vd &c)
35   : m(b.size()), n(c.size()), N(n + 1), B(m),
36     D(m + 2, vd(n + 2)) {
37     for (int i = 0; i < m; i++)
38       for (int j = 0; j < n; j++) D[i][j] = A[i][j];
39     for (int i = 0; i < m; i++) {
40       B[i] = n + i;
41       D[i][n] = -1;
42       D[i][n + 1] = b[i];
43     }
44     for (int j = 0; j < n; j++) {
45       N[j] = j;
46       D[m][j] = -c[j];
47     }
48     N[n] = -1;
49     D[m + 1][n] = 1;
50   }
51
52   void Pivot(int r, int s) {
53     double inv = 1.0 / D[r][s];
54     for (int i = 0; i < m + 2; i++)
55       if (i != r)

```

```

57       if (j != s) D[i][j] -= D[r][j] * D[i][s] * inv;
58     for (int j = 0; j < n + 2; j++)
59       if (j != s) D[r][j] *= inv;
60     for (int i = 0; i < m + 2; i++)
61       if (i != r) D[i][s] *= -inv;
62     D[r][s] = inv;
63     swap(B[r], N[s]);
64   }
65
66   bool Simplex(int phase) {
67     int x = phase == 1 ? m + 1 : m;
68     while (true) {
69       int s = -1;
70       for (int j = 0; j <= n; j++) {
71         if (phase == 2 && N[j] == -1) continue;
72         if (s == -1 || D[x][j] < D[x][s] ||
73             D[x][j] == D[x][s] && N[j] < N[s])
74           s = j;
75       }
76       if (D[x][s] > -EPS) return true;
77       int r = -1;
78       for (int i = 0; i < m; i++) {
79         if (D[i][s] < EPS) continue;
80         if (r == -1 ||
81             D[i][n + 1] / D[i][s] < D[r][n + 1] / D[r][s] ||
82             (D[i][n + 1] / D[i][s]) ==
83             (D[r][n + 1] / D[r][s]) &&
84             B[i] < B[r])
85           r = i;
86       }
87       if (r == -1) return false;
88       Pivot(r, s);
89     }
90
91   ld Solve(vd &x) {
92     int r = 0;
93     for (int i = 1; i < m; i++)
94       if (D[i][n + 1] < D[r][n + 1]) r = i;
95     if (D[r][n + 1] < -EPS) {
96       Pivot(r, n);
97       if (!Simplex(1) || D[m + 1][n + 1] < -EPS)
98         return numeric_limits<ld>::infinity();
99       for (int i = 0; i < m; i++)
100         if (B[i] == -1) {
101           int s = -1;
102           for (int j = 0; j <= n; j++)
103             if (s == -1 || D[i][j] < D[i][s] ||
104                 D[i][j] == D[i][s] && N[j] < N[s])
105               s = j;
106           Pivot(i, s);
107         }
108       if (!Simplex(2)) return numeric_limits<ld>::infinity();
109       x = vd(n);
110       for (int i = 0; i < m; i++)
111         if (B[i] < n) x[B[i]] = D[i][n + 1];
112       return D[m][n + 1];
113     }
114   };
115
116   int main() {
117     const int m = 4;
118     const int n = 3;
119     ld _A[m][n] = {
120       {6, -1, 0}, {-1, -5, 0}, {1, 5, 1}, {-1, -5, -1}};
121     ld _b[m] = {10, -4, 5, -5};
122     ld _c[n] = {1, -1, 0};
123
124     vvd A(m);
125     vd b(_b, _b + m);
126     vd c(_c, _c + n);
127     for (int i = 0; i < m; i++) A[i] = vd(_A[i], _A[i] + n);
128
129     LPSolver solver(A, b, c);
130     vd x;
131     ld value = solver.Solve(x);
132
133     cerr << "VALUE: " << value << endl; // VALUE: 1.29032
134     cerr << "SOLUTION:"; // SOLUTION: 1.74194 0.451613 1
135     for (size_t i = 0; i < x.size(); i++) cerr << " " << x[i];
136     cerr << endl;
137     return 0;
138   }
}

```

6. Geometry

6.1. Point

```

1 template <typename T> struct P {
2   T x, y;
3   P(T x = 0, T y = 0) : x(x), y(y) {}

```

```

5   bool operator<(const P &p) const {
6     return tie(x, y) < tie(p.x, p.y);
7   }
8   bool operator==(const P &p) const {
9     return tie(x, y) == tie(p.x, p.y);
10  }
11  P operator-() const { return {-x, -y}; }
12  P operator+(P p) const { return {x + p.x, y + p.y}; }
13  P operator-(P p) const { return {x - p.x, y - p.y}; }
14  P operator*(T d) const { return {x * d, y * d}; }
15  P operator/(T d) const { return {x / d, y / d}; }
16  T dist2() const { return x * x + y * y; }
17  double len() const { return sqrt(dist2()); }
18  P unit() const { return *this / len(); }
19  friend T dot(P a, P b) { return a.x * b.x + a.y * b.y; }
20  friend T cross(P a, P b) { return a.x * b.y - a.y * b.x; }
21  friend T cross(P a, P b, P o) {
22    return cross(a - o, b - o);
23  }
24}; using pt = P<ll>;

```

6.1.1. Quaternion

```

1 constexpr double PI = 3.141592653589793;
2 constexpr double EPS = 1e-7;
3 struct Q {
4   using T = double;
5   T x, y, z, r;
6   Q(T r = 0) : x(0), y(0), z(0), r(r) {}
7   Q(T x, T y, T z, T r = 0) : x(x), y(y), z(z), r(r) {}
8   friend bool operator==(const Q &a, const Q &b) {
9     return (a - b).abs2() <= EPS;
10  }
11  friend bool operator!=(const Q &a, const Q &b) {
12    return !(a == b);
13  }
14  Q operator-() { return Q(-x, -y, -z, -r); }
15  Q operator+(const Q &b) const {
16    return Q(x + b.x, y + b.y, z + b.z, r + b.r);
17  }
18  Q operator-(const Q &b) const {
19    return Q(x - b.x, y - b.y, z - b.z, r - b.r);
20  }
21  Q operator*(const T &t) const {
22    return Q(x * t, y * t, z * t, r * t);
23  }
24  Q operator*(const Q &b) const {
25    return Q(r * b.x + x * b.r + y * b.z - z * b.y,
26            r * b.y - x * b.z + y * b.r + z * b.x,
27            r * b.z + x * b.y - y * b.x + z * b.r,
28            r * b.r - x * b.x - y * b.y - z * b.z);
29  }
30  Q operator/(const Q &b) const { return *this * b.inv(); }
31  T abs2() const { return r * r + x * x + y * y + z * z; }
32  T len() const { return sqrt(abs2()); }
33  Q conj() const { return Q(-x, -y, -z, r); }
34  Q unit() const { return *this * (1.0 / len()); }
35  Q inv() const { return conj() * (1.0 / abs2()); }
36  friend T dot(Q a, Q b) {
37    return a.x * b.x + a.y * b.y + a.z * b.z;
38  }
39  friend Q cross(Q a, Q b) {
40    return Q(a.y * b.z - a.z * b.y, a.z * b.x - a.x * b.z,
41             a.x * b.y - a.y * b.x);
42  }
43  friend Q rotation_around(Q axis, T angle) {
44    return axis.unit() * sin(angle / 2) + cos(angle / 2);
45  }
46  Q rotated_around(Q axis, T angle) {
47    Q u = rotation_around(axis, angle);
48    return u * *this / u;
49  }
50  friend Q rotation_between(Q a, Q b) {
51    a = a.unit(), b = b.unit();
52    if (a == -b) {
53      // degenerate case
54      Q ortho = abs(a.y) > EPS ? cross(a, Q(1, 0, 0))
55                                : cross(a, Q(0, 1, 0));
56      return rotation_around(ortho, PI);
57    }
58    return (a * (a + b)).conj();
59  }

```

6.1.2. Spherical Coordinates

```

1 struct car_p {
2   double x, y, z;
3 };
4 struct sph_p {
5   double r, theta, phi;
6 };

```

```

7   sph_p conv(car_p p) {
8     double r = sqrt(p.x * p.x + p.y * p.y + p.z * p.z);
9     double theta = asin(p.y / r);
10    double phi = atan2(p.y, p.x);
11    return {r, theta, phi};
12  }
13  car_p conv(sph_p p) {
14    double x = p.r * cos(p.theta) * sin(p.phi);
15    double y = p.r * cos(p.theta) * cos(p.phi);
16    double z = p.r * sin(p.theta);
17    return {x, y, z};
18  }
19 }

```

6.2. Segments

```

1 // for non-collinear ABCD, if segments AB and CD intersect
2 bool intersects(pt a, pt b, pt c, pt d) {
3   if (cross(b, c, a) * cross(b, d, a) > 0) return false;
4   if (cross(d, a, c) * cross(d, b, c) > 0) return false;
5   return true;
6 }
7 // the intersection point of lines AB and CD
8 pt intersect(pt a, pt b, pt c, pt d) {
9   auto x = cross(b, c, a), y = cross(b, d, a);
10  if (x == y) {
11    // if(abs(x, y) < 1e-8) {
12    // is parallel
13    } else {
14      return d * (x / (x - y)) - c * (y / (x - y));
15    }
16  }
17 }

```

6.3. Convex Hull

```

1 // returns a convex hull in counterclockwise order
2 // for a non-strict one, change cross >= to >
3 vector<pt> convex_hull(vector<pt> p) {
4   sort(ALL(p));
5   if (p[0] == p.back()) return {p[0]};
6   int n = p.size(), t = 0;
7   vector<pt> h(n + 1);
8   for (int _ = 2, s = 0; _--; s = --t, reverse(ALL(p)))
9     for (pt i : p) {
10       while (t > s + 1 && cross(i, h[t - 1], h[t - 2]) >= 0)
11         t--;
12       h[t++] = i;
13     }
14   return h.resize(t), h;
15 }

```

6.3.1. 3D Hull

```

1
2 3 typedef Point3D<double> P3;
4
5 4 struct PR {
6    void ins(int x) { (a == -1 ? a : b) = x; }
7    void rem(int x) { (a == x ? a : b) = -1; }
8    int cnt() { return (a != -1) + (b != -1); }
9    int a, b;
10  };
11
12 4 struct F {
13   P3 q;
14   int a, b, c;
15  };
16
17 4 vector<F> hull3d(const vector<P3> &A) {
18   assert(sz(A) >= 4);
19   vector<vector<PR>> E(sz(A)), vector<PR>(sz(A), {-1, -1});
20   #define E(x, y) E[f.x][f.y]
21   vector<F> FS;
22   auto mf = [&](int i, int j, int k, int l) {
23     P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
24     if (q.dot(A[l]) > q.dot(A[i])) q = q * -1;
25     F f{q, i, j, k};
26     E(a, b).ins(k);
27     E(a, c).ins(j);
28     E(b, c).ins(i);
29     FS.push_back(f);
30   };
31   rep(i, 0, 4) rep(j, i + 1, 4) rep(k, j + 1, 4)
32   mf(i, j, k, 6 - i - j - k);
33
34   rep(i, 4, sz(A)) {
35     rep(j, 0, sz(FS)) {
36       F f = FS[j];
37       if (f.q.dot(A[i]) > f.q.dot(A[f.a])) {
38         E(a, b).rem(f.c);
39         E(a, c).rem(f.b);
40         E(b, c).rem(f.a);
41       }
42     }
43   }
44 }

```

```

41     swap(FS[j--], FS.back());
42     FS.pop_back();
43   }
44   int nw = sz(FS);
45   rep(j, 0, nw) {
46     F f = FS[j];
47   #define C(a, b, c)
48   if (E(a, b).cnt() != 2) mf(f.a, f.b, i, f.c);
49     C(a, b, c);
50     C(a, c, b);
51     C(b, c, a);
52   }
53 }
54 for (F &it : FS)
55   if ((A[it.b] - A[it.a])
56       .cross(A[it.c] - A[it.a])
57       .dot(it.q) <= 0)
58     swap(it.c, it.b);
59 return FS;
60 };
61 };

```

6.4. Angular Sort

```

1 auto angle_cmp = [](const pt &a, const pt &b) {
2   auto btm = [](const pt &a) {
3     return a.y < 0 || (a.y == 0 && a.x < 0);
4   };
5   return make_tuple(btm(a), a.y * b.x, abs2(a)) <
6         make_tuple(btm(b), a.x * b.y, abs2(b));
7 };
8 void angular_sort(vector<pt> &p) {
9   sort(p.begin(), p.end(), angle_cmp);
10 }

```

6.5. Convex Polygon Minkowski Sum

```

1 // O(n) convex polygon minkowski sum
2 // must be sorted and counterclockwise
3 vector<pt> minkowski_sum(vector<pt> p, vector<pt> q) {
4   auto diff = [](vector<pt> &c) {
5     auto rcmp = [](pt a, pt b) {
6       return pt{a.y, a.x} < pt{b.y, b.x};
7     };
8     rotate(c.begin(), min_element(ALL(c), rcmp), c.end());
9     c.push_back(c[0]);
10    vector<pt> ret;
11    for (int i = 1; i < c.size(); i++)
12      ret.push_back(c[i] - c[i - 1]);
13    return ret;
14  };
15  auto dp = diff(p), dq = diff(q);
16  pt cur = p[0] + q[0];
17  vector<pt> d(dp.size() + dq.size()), ret = {cur};
18  // include angle_cmp from angular-sort.cpp
19  merge(ALL(dp), ALL(dq), d.begin(), angle_cmp);
20  // optional: make ret strictly convex (UB if degenerate)
21  int now = 0;
22  for (int i = 1; i < d.size(); i++) {
23    if (cross(d[i], d[now]) == 0) d[now] = d[now] + d[i];
24    else d[++now] = d[i];
25  }
26  d.resize(now + 1);
27  // end optional part
28  for (pt v : d) ret.push_back(cur = cur + v);
29  return ret.pop_back(), ret;
30 }

```

6.6. Point In Polygon

```

1 bool on_segment(pt a, pt b, pt p) {
2   return cross(a, b, p) == 0 && dot((p - a), (p - b)) <= 0;
3 }
4 // p can be any polygon, but this is O(n)
5 bool inside(const vector<pt> &p, pt a) {
6   int cnt = 0, n = p.size();
7   for (int i = 0; i < n; i++) {
8     pt l = p[i], r = p[(i + 1) % n];
9     // change to return 0; for strict version
10    if (on_segment(l, r, a)) return 1;
11    cnt ^= ((a.y < l.y) - (a.y < r.y)) * cross(l, r, a) > 0;
12  }
13  return cnt;
14 }

```

6.6.1. Convex Version

```

1 // no preprocessing version
2 // p must be a strict convex hull, counterclockwise
3 // if point is inside or on border
4 bool is_inside(const vector<pt> &c, pt p) {
5   int n = c.size(), l = 1, r = n - 1;
6   if (cross(c[0], c[1], p) < 0) return false;
7 }

```

```

7   if (cross(c[n - 1], c[0], p) < 0) return false;
8   while (l < r - 1) {
9     int m = (l + r) / 2;
10    T a = cross(c[0], c[m], p);
11    if (a > 0) l = m;
12    else if (a < 0) r = m;
13    else return dot(c[0] - p, c[m] - p) <= 0;
14  }
15  if (l == r) return dot(c[0] - p, c[l] - p) <= 0;
16  else return cross(c[l], c[r], p) >= 0;
17 }

19 // with preprocessing version
20 vector<pt> vecs;
21 pt center;
22 // p must be a strict convex hull, counterclockwise
23 // BEWARE OF OVERFLOWS!
24 void preprocess(vector<pt> p) {
25   for (auto &v : p) v = v * 3;
26   center = p[0] + p[1] + p[2];
27   center.x /= 3, center.y /= 3;
28   for (auto &v : p) v = v - center;
29   vecs = (angular_sort(p), p);
30 }

31 bool intersect_strict(pt a, pt b, pt c, pt d) {
32   if (cross(b, c, a) * cross(b, d, a) > 0) return false;
33   if (cross(d, a, c) * cross(d, b, c) >= 0) return false;
34   return true;
35 }
36 // if point is inside or on border
37 bool query(pt p) {
38   p = p * 3 - center;
39   auto pr = upper_bound(ALL(vecs), p, angle_cmp);
40   if (pr == vecs.end()) pr = vecs.begin();
41   auto pl = (pr == vecs.begin()) ? vecs.back() : *(pr - 1);
42   return !intersect_strict({0, 0}, p, pl, *pr);
43 }

```

6.6.2. Offline Multiple Points Version

Requires: Point, GNU PBDS

```

1
2
3
4
5 using Double = __float128;
6 using Point = pt<Double, Double>;
7
8 int n, m;
9 vector<Point> poly;
10 vector<Point> query;
11 vector<int> ans;
12
13 struct Segment {
14   Point a, b;
15   int id;
16 };
17 vector<Segment> segs;
18
19 Double Xnow;
20 inline Double get_y(const Segment &u, Double xnow = Xnow) {
21   const Point &a = u.a;
22   const Point &b = u.b;
23   return (a.y * (b.x - xnow) + b.y * (xnow - a.x)) /
24         (b.x - a.x);
25 }
26
27 bool operator<(Segment u, Segment v) {
28   Double yu = get_y(u);
29   Double yv = get_y(v);
30   if (yu != yv) return yu < yv;
31   return u.id < v.id;
32 }
33 ordered_map<Segment> st;
34
35 struct Event {
36   int type; // +1 insert seg, -1 remove seg, 0 query
37   Double x, y;
38   int id;
39 };
40
41 bool operator<(Event a, Event b) {
42   if (a.x != b.x) return a.x < b.x;
43   if (a.type != b.type) return a.type < b.type;
44   return a.y < b.y;
45 }
46 vector<Event> events;
47
48 void solve() {
49   set<Double> xs;
50   set<Point> ps;
51   for (int i = 0; i < n; i++) {
52     xs.insert(poly[i].x);
53     ps.insert(poly[i]);
54   }
55 }

```

```

53 }
54 for (int i = 0; i < n; i++) {
55     Segment s{poly[i], poly[(i + 1) % n], i};
56     if (s.a.x > s.b.x ||
57         (s.a.x == s.b.x && s.a.y > s.b.y)) {
58         swap(s.a, s.b);
59     }
60     segs.push_back(s);
61
62     if (s.a.x != s.b.x) {
63         events.push_back({+1, s.a.x + 0.2, s.a.y, i});
64         events.push_back({-1, s.b.x - 0.2, s.b.y, i});
65     }
66 }
67 for (int i = 0; i < m; i++) {
68     events.push_back({0, query[i].x, query[i].y, i});
69 }
70 sort(events.begin(), events.end());
71 int cnt = 0;
72 for (Event e : events) {
73     int i = e.id;
74     Xnow = e.x;
75     if (e.type == 0) {
76         Double x = e.x;
77         Double y = e.y;
78         Segment tmp = {{x - 1, y}, {x + 1, y}, -1};
79         auto it = st.lower_bound(tmp);
80
81         if (ps.count(query[i]) > 0) {
82             ans[i] = 0;
83         } else if (xs.count(x) > 0) {
84             ans[i] = -2;
85         } else if (it != st.end() &&
86                     get_y(*it) == get_y(tmp)) {
87             ans[i] = 0;
88         } else if (it != st.begin() &&
89                     get_y(*prev(it)) == get_y(tmp)) {
90             ans[i] = 0;
91         } else {
92             int rk = st.order_of_key(tmp);
93             if (rk % 2 == 1) {
94                 ans[i] = 1;
95             } else {
96                 ans[i] = -1;
97             }
98         }
99     } else if (e.type == 1) {
100        st.insert(segs[i]);
101        assert((int)st.size() == ++cnt);
102    } else if (e.type == -1) {
103        st.erase(segs[i]);
104        assert((int)st.size() == --cnt);
105    }
106 }
107 }
```

6.7. Closest Pair

```

1 vector<pll> p; // sort by x first!
2 bool cmpy(const pll &a, const pll &b) const {
3     return a.y < b.y;
4 }
5 ll sq(ll x) { return x * x; }
6 // returns (minimum dist)^2 in [l, r)
7 ll solve(int l, int r) {
8     if (r - l <= 1) return 1e18;
9     int m = (l + r) / 2;
10    ll mid = p[m].x, d = min(solve(l, m), solve(m, r));
11    auto pb = p.begin();
12    inplace_merge(pb + l, pb + m, pb + r, cmpy);
13    vector<pll> s;
14    for (int i = l; i < r; i++)
15        if (sq(p[i].x - mid) < d) s.push_back(p[i]);
16    for (int i = 0; i < s.size(); i++)
17        for (int j = i + 1;
18             j < s.size() && sq(s[j].y - s[i].y) < d; j++)
19            d = min(d, dis(s[i], s[j]));
20    return d;
21 }
```

6.8. Minimum Enclosing Circle

```

1
2 typedef Point<double> P;
3 double ccRadius(const P &A, const P &B, const P &C) {
4     return (B - A).dist() * (C - B).dist() * (A - C).dist() /
5            abs((B - A).cross(C - A)) / 2;
6 }
7 P ccCenter(const P &A, const P &B, const P &C) {
8     P b = C - A, c = B - A;
9     return A + (b * c.dist2() - c * b.dist2()).perp() /
10            b.cross(c) / 2;
11 }
```

```

13 pair<P, double> mec(vector<P> ps) {
14     shuffle(all(ps), mt19937(time(0)));
15     P o = ps[0];
16     double r = 0, EPS = 1 + 1e-8;
17     rep(i, 0, sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
18         o = ps[i], r = 0;
19         rep(j, 0, i) if ((o - ps[j]).dist() > r * EPS) {
20             o = (ps[i] + ps[j]) / 2;
21             r = (o - ps[i]).dist();
22             rep(k, 0, j) if ((o - ps[k]).dist() > r * EPS) {
23                 o = ccCenter(ps[i], ps[j], ps[k]);
24                 r = (o - ps[i]).dist();
25             }
26         }
27     }
28     return {o, r};
29 }
```

6.9. Delaunay Triangulation

```

1
2 typedef Point<ll> P;
3 typedef struct Quad *Q;
4 typedef __int128_t lll; // (can be ll if coords are < 2e4)
5 P arb(LLONG_MAX, LLONG_MAX); // not equal to any other point
6
7 struct Quad {
8     bool mark;
9     Q o, rot;
10    P p;
11    P F() { return r()>p; }
12    Q r() { return rot->rot; }
13    Q prev() { return rot->o->rot; }
14    Q next() { return r()->prev(); }
15 };
16
17 bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
18     lll p2 = p.dist2(), A = a.dist2() - p2,
19                      B = b.dist2() - p2, C = c.dist2() - p2;
20     return p.cross(a, b) * C + p.cross(b, c) * A +
21            p.cross(c, a) * B >
22            0;
23 }
24 Q makeEdge(P orig, P dest) {
25     Q q[] = {new Quad{0, 0, 0, orig}, new Quad{0, 0, 0, arb},
26              new Quad{0, 0, 0, dest}, new Quad{0, 0, 0, arb}};
27     rep(i, 0, 4) q[i]->o = q[-i & 3],
28                  q[i]->rot = q[(i + 1) & 3];
29     return *q;
30 }
31 void splice(Q a, Q b) {
32     swap(a->o->rot->o, b->o->rot->o);
33     swap(a->o, b->o);
34 }
35 Q connect(Q a, Q b) {
36     Q q = makeEdge(a->F(), b->p);
37     splice(q, a->next());
38     splice(q->r(), b);
39     return q;
40 }
41
42 pair<Q, Q> rec(const vector<P> &s) {
43     if (sz(s) <= 3) {
44         Q a = makeEdge(s[0], s[1]),
45             b = makeEdge(s[1], s.back());
46         if (sz(s) == 2) return {a, a->r()};
47         splice(a->r(), b);
48         auto side = s[0].cross(s[1], s[2]);
49         Q c = side ? connect(b, a) : 0;
50         return {side < 0 ? c->r() : a, side < 0 ? c : b->r()};
51     }
52
53 #define H(e) e->F(), e->p
54 #define valid(e) (e->F().cross(H(base)) > 0)
55     Q A, B, ra, rb;
56     int half = sz(s) / 2;
57     tie(ra, A) = rec({all(s) - half});
58     tie(B, rb) = rec({sz(s) - half + all(s)}));
59     while ((B->p.cross(H(A)) < 0 && (A = A->next())) ||
60            (A->p.cross(H(B)) > 0 && (B = B->r()->o)));
61     Q base = connect(B->r(), A);
62     if (A->p == ra->p) ra = base->r();
63     if (B->p == rb->p) rb = base;
64
65 #define DEL(e, init, dir)
66 Q e = init->dir;
67 if (valid(e))
68     while (circ(e->dir->F(), H(base), e->F())) {
69         Q t = e->dir;
70         splice(e, e->prev());
71         splice(e->r(), e->r()->prev());
72         e = t;
73 }
```

```

75
76     }
77
78     for (;;) {
79         DEL(LC, base->r(), o);
80         DEL(RC, base, prev());
81         if (!valid(LC) && !valid(RC)) break;
82         if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
83             base = connect(RC, base->r());
84         else base = connect(base->r(), LC->r());
85     }
86     return {ra, rb};
87 }
88
89 // returns [A_0, B_0, C_0, A_1, B_1, ...]
90 // where A_i, B_i, C_i are counter-clockwise triangles
91 vector<P> triangulate(vector<P> pts) {
92     sort(all(pts));
93     assert(unique(all(pts)) == pts.end());
94     if (sz(pts) < 2) return {};
95     Q e = rec(pts).first;
96     vector<Q> q = {e};
97     int qi = 0;
98     while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
99 #define ADD
100 {
101     Q c = e;
102     do {
103         c->mark = 1;
104         pts.push_back(c->p);
105         q.push_back(c->r());
106         c = c->next();
107     } while (c != e);
108 }
109 ADD;
110 pts.clear();
111 while (qi < sz(q))
112     if (!(e = q[qi++])->mark) ADD;
113 return pts;
114 }
```

```
33     p[last - 1] = GetIntersection(q[last - 1], q[last]);
34 }
35 while (first < last && !OnLeft(q[first], p[last - 1]))
36     last--;
37 if (last - first <= 1) return 0;
38 p[last] = GetIntersection(q[last], q[first]);
39
40 int m = 0;
41 for (int i = first; i <= last; i++) poly[m++] = p[i];
42 return m;
43 }
```

7. Strings

7.1. Knuth-Morris-Pratt Algorithm

```
1
3 vector<int> pi(const string &s) {
4     vector<int> p(s.size());
5     for (int i = 1; i < s.size(); i++) {
6         int g = p[i - 1];
7         while (g && s[i] != s[g]) g = p[g - 1];
8         p[i] = g + (s[i] == s[g]);
9     }
10    return p;
11}
12 vector<int> match(const string &s, const string &pat) {
13    vector<int> p = pi(pat + '\0' + s), res;
14    for (int i = p.size() - s.size(); i < p.size(); i++)
15        if (p[i] == pat.size())
16            res.push_back(i - 2 * pat.size());
17    return res;
18}
```

7.2. Aho-Corasick Automaton

```

1 struct Aho_Corasick {
2     static const int maxc = 26, maxn = 4e5;
3     struct NODES {
4         int Next[maxc], fail, ans;
5     };
6     NODES T[maxn];
7     int top, qtop, q[maxn];
8     int get_node(const int &fail) {
9         fill_n(T[top].Next, maxc, 0);
10        T[top].fail = fail;
11        T[top].ans = 0;
12        return top++;
13    }
14    int insert(const string &s) {
15        int ptr = 1;
16        for (char c : s) { // change char id
17            c -= 'a';
18            if (!T[ptr].Next[c]) T[ptr].Next[c] = get_node(ptr);
19            ptr = T[ptr].Next[c];
20        }
21        return ptr;
22    } // return ans_last_place
23    void build_fail(int ptr) {
24        int tmp;
25        for (int i = 0; i < maxc; i++) {
26            if (T[ptr].Next[i]) {
27                tmp = T[ptr].fail;
28                while (tmp != 1 && !T[tmp].Next[i])
29                    tmp = T[tmp].fail;
30                if (T[tmp].Next[i] != T[ptr].Next[i])
31                    if (T[tmp].Next[i]) tmp = T[tmp].Next[i];
32                T[ptr].Next[i].fail = tmp;
33                q[qtop++] = T[ptr].Next[i];
34            }
35        }
36    void AC_auto(const string &s) {
37        int ptr = 1;
38        for (char c : s) {
39            while (ptr != 1 && !T[ptr].Next[c]) ptr = T[ptr].fail;
40            if (T[ptr].Next[c]) {
41                ptr = T[ptr].Next[c];
42                T[ptr].ans++;
43            }
44        }
45    }
46    void Solve(string &s) {
47        for (char &c : s) // change char id
48            c -= 'a';
49        for (int i = 0; i < qtop; i++) build_fail(q[i]);
50        AC_auto(s);
51        for (int i = qtop - 1; i > -1; i--)
52            T[T[q[i]].fail].ans += T[q[i]].ans;
53    }
54    void reset() {
55        qtop = top = q[0] = 1;

```

6.9.1. Slower Version

```

1
3 template <class P, class F>
4 void delaunay(vector<P> &ps, F trifun) {
5     if (sz(ps) == 3) {
6         int d = (ps[0].cross(ps[1], ps[2]) < 0);
7         trifun(0, 1 + d, 2 - d);
8     }
9     vector<P3> p3;
10    for (P p : ps) p3.emplace_back(p.x, p.y, p.dist2());
11    if (sz(ps) > 3)
12        for (auto t : hull3d(p3))
13            if ((p3[t.b] - p3[t.a])
14                .cross(p3[t.c] - p3[t.a])
15                .dot(P3(0, 0, 1)) < 0)
16                trifun(t.a, t.c, t.b);
17 }

```

6.10. Half Plane Intersection

```

1 struct Line {
2     Point P;
3     Vector v;
4     bool operator<(const Line &b) const {
5         return atan2(v.y, v.x) < atan2(b.v.y, b.v.x);
6     }
7 };
8 bool OnLeft(const Line &L, const Point &p) {
9     return Cross(L.v, p - L.P) > 0;
10 }
11 Point GetIntersection(Line a, Line b) {
12     Vector u = a.P - b.P;
13     Double t = Cross(b.v, u) / Cross(a.v, b.v);
14     return a.P + a.v * t;
15 }
16 int HalfplaneIntersection(Line *L, int n, Point *poly) {
17     sort(L, L + n);
18
19     int first, last;
20     Point *p = new Point[n];
21     Line *q = new Line[n];
22     q[first = last = 0] = L[0];
23     for (int i = 1; i < n; i++) {
24         while (first < last && !OnLeft(L[i], p[last - 1]))
25             last--;
26         while (first < last && !OnLeft(L[i], p[first])) first++;
27         q[++last] = L[i];
28         if (fabs(Cross(q[last].v, q[last - 1].v)) < EPS) {
29             last--;
30             if (OnLeft(q[last], L[i].P)) q[last] = L[i];
31         }
32     }
33     if (first < last)
34         sort(q, q + last);
35     else
36         delete[] q;
37     delete[] p;
38     return last;
39 }

```

```

57     get_node(1);
58 }
59 // usage example
60 string s, S;
61 int n, t, ans_place[50000];
62 int main() {
63     Tie cin >> t;
64     while (t--) {
65         AC.reset();
66         cin >> S >> n;
67         for (int i = 0; i < n; i++) {
68             cin >> s;
69             ans_place[i] = AC.insert(s);
70         }
71     AC.Solve(S);
72     for (int i = 0; i < n; i++)
73         cout << AC.T[ans_place[i]].ans << '\n';
74 }

```

7.3. Suffix Array

```

1
3
4 // sa[i]: starting index of suffix at rank i
5 //          0-indexed, sa[0] = n (empty string)
6 // lcp[i]: lcp of sa[i] and sa[i - 1], lcp[0] = 0
7 struct SuffixArray {
8     vector<int> sa, lcp;
9     SuffixArray(string &s,
10                int lim = 256) { // or basic_string<int>
11        int n = sz(s) + 1, k = 0, a, b;
12        vector<int> x(all(s) + 1), y(n), ws(max(n, lim)),
13        rank(n);
14        sa = lcp = y, iota(all(sa), 0);
15        for (int j = 0, p = 0; p < n;
16             j = max(1, j * 2), lim = p) {
17            p = j, iota(all(y), n - j);
18            for (int i = 0; i < n; i++)
19                if (sa[i] >= j) y[p++] = sa[i] - j;
20            fill(all(ws), 0);
21            for (int i = 0; i < n; i++) ws[x[i]]++;
22            for (int i = 1; i < lim; i++) ws[i] += ws[i - 1];
23            for (int i = n; i--;) sa[-ws[x[y[i]]]] = y[i];
24            swap(x, y), p = 1, x[sa[0]] = 0;
25            for (int i = 1; i < n; i++)
26                a = sa[i - 1], b = sa[i],
27                x[b] = (y[a] == y[b] && y[a + j] == y[b + j])
28                    ? p - 1 : p++;
29
30        }
31        for (int i = 1; i < n; i++) rank[sa[i]] = i;
32        for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
33            for (k &=&k--, j = sa[rank[i] - i];
34                 s[i + k] == s[j + k]; k++);
35    }
36 };

```

7.4. Suffix Tree

```

1 struct SAM {
2     static const int maxc = 26; // char range
3     static const int maxn = 10010; // string len
4     struct Node {
5         Node *green, *edge[maxc];
6         int max_len, in, times;
7     } *root, *last, reg[maxn * 2];
8     int top;
9     Node *get_node(int _max) {
10        Node *re = &reg[top++];
11        re->in = 0, re->times = 1;
12        re->max_len = _max, re->green = 0;
13        for (int i = 0; i < maxc; i++) re->edge[i] = 0;
14        return re;
15    }
16    void insert(const char c) { // c in range [0, maxc)
17        Node *p = last;
18        last = get_node(p->max_len + 1);
19        while (p && !p->edge[c])
20            p->edge[c] = last, p = p->green;
21        if (!p) last->green = root;
22        else {
23            Node *pot_green = p->edge[c];
24            if ((pot_green->max_len) == (p->max_len + 1))
25                last->green = pot_green;
26            else {
27                Node *wish = get_node(p->max_len + 1);
28                wish->times = 0;
29                while (p && p->edge[c] == pot_green)
30                    p->edge[c] = wish, p = p->green;
31            }
32        }
33    }
34 };

```

```

31     for (int i = 0; i < maxc; i++)
32         wish->edge[i] = pot_green->edge[i];
33     wish->green = pot_green->green;
34     pot_green->green = wish;
35     last->green = wish;
36 }
37 }
38 Node *q[maxn * 2];
39 int ql, qr;
40 void get_times(Node *p) {
41     ql = 0, qr = -1, reg[0].in = 1;
42     for (int i = 1; i < top; i++) reg[i].green->in++;
43     for (int i = 0; i < top; i++)
44         if (!reg[i].in) q[++qr] = &reg[i];
45     while (ql <= qr) {
46         q[ql]->green->times += q[ql]->times;
47         if (!(--q[ql]->green->in)) q[++qr] = q[ql]->green;
48         ql++;
49     }
50 }
51 void build(const string &s) {
52     top = 0;
53     root = last = get_node(0);
54     for (char c : s) insert(c - 'a'); // change char id
55     get_times(root);
56 }
57 // call build before solve
58 int solve(const string &s) {
59     Node *p = root;
60     for (char c : s)
61         if (!(p = p->edge[c - 'a'])) // change char id
62             return 0;
63     return p->times;
64 }
65 };

```

7.5. Cocke-Younger-Kasami Algorithm

```

1
2
3 struct rule {
4     // s -> xy
5     // if y == -1, then s -> x (unit rule)
6     int s, x, y, cost;
7 };
8 int state;
9 // state (id) for each letter (variable)
10 // lowercase letters are terminal symbols
11 map<char, int> rules;
12 vector<rule> cnf;
13 void init() {
14     state = 0;
15     rules.clear();
16     cnf.clear();
17 }
18 // convert a cfg rule to cnf (but with unit rules) and add
19 // it
20 void add_to_cnf(char s, const string &p, int cost) {
21     if (!rules.count(s)) rules[s] = state++;
22     for (char c : p)
23         if (!rules.count(c)) rules[c] = state++;
24     if (p.size() == 1) {
25         cnf.push_back({rules[s], rules[p[0]], -1, cost});
26     } else {
27         // length >= 3 -> split
28         int left = rules[s];
29         int sz = p.size();
30         for (int i = 0; i < sz - 2; i++) {
31             cnf.push_back({left, rules[p[i]], state, 0});
32             left = state++;
33         }
34         cnf.push_back(
35             {left, rules[p[sz - 2]], rules[p[sz - 1]], cost});
36     }
37 }
38
39 constexpr int MAXN = 55;
40 vector<long long> dp[MAXN][MAXN];
41 // unit rules with negative costs can cause negative cycles
42 vector<bool> neg_INF[MAXN][MAXN];
43
44 void relax(int l, int r, rule c, long long cost,
45            bool neg_c = 0) {
46     if (!neg_INF[l][r][c.s] &&
47         (neg_INF[l][r][c.x] || cost < dp[l][r][c.s])) {
48         if (neg_c || neg_INF[l][r][c.x]) {
49             dp[l][r][c.s] = 0;
50             neg_INF[l][r][c.s] = true;
51         } else {
52             dp[l][r][c.s] = cost;
53         }
54     }
55 }

```

```

57 void bellman(int l, int r, int n) {
58     for (int k = 1; k <= state; k++)
59         for (rule c : cnf)
60             if (c.y == -1)
61                 relax(l, r, c, dp[l][r][c.x] + c.cost, k == n);
62 }
63 void cyk(const string &s) {
64     vector<int> tok;
65     for (char c : s) tok.push_back(rules[c]);
66     for (int i = 0; i < tok.size(); i++) {
67         for (int j = 0; j < tok.size(); j++) {
68             dp[i][j] = vector<long long>(state + 1, INT_MAX);
69             neg_INF[i][j] = vector<bool>(state + 1, false);
70         }
71         dp[i][i][tok[i]] = 0;
72         bellman(i, i, tok.size());
73     }
74     for (int r = 1; r < tok.size(); r++) {
75         for (int l = r - 1; l >= 0; l--) {
76             for (int k = l; k < r; k++)
77                 for (rule c : cnf)
78                     if (c.y != -1)
79                         relax(l, r, c,
80                               dp[l][k][c.x] + dp[k + 1][r][c.y] +
81                               c.cost);
82         bellman(l, r, tok.size());
83     }
84 }
85 // usage example
86 int main() {
87     init();
88     add_to_cnf('S', "aSc", 1);
89     add_to_cnf('S', "BBB", 1);
90     add_to_cnf('S', "SB", 1);
91     add_to_cnf('B', "b", 1);
92     cyk("abbbbc");
93     // dp[0][s.size() - 1][rules[start]] = min cost to
94     // generate s
95     cout << dp[0][5][rules['S']] << '\n'; // 7
96     cyk("acbc");
97     cout << dp[0][3][rules['S']] << '\n'; // INT_MAX
98     add_to_cnf('S', "S", -1);
99     cyk("abbbbc");
100    cout << neg_INF[0][5][rules['S']] << '\n'; // 1
101 }

```

7.6. Z Value

```

1 int z[n];
2 void zval(string s) {
3     // z[i] => longest common prefix of s and s[i:], i > 0
4     int n = s.size();
5     z[0] = 0;
6     for (int b = 0, i = 1; i < n; i++) {
7         if (z[b] + b <= i) z[i] = 0;
8         else z[i] = min(z[i - b], z[b] + b - i);
9         while (s[i + z[i]] == s[z[i]]) z[i]++;
10        if (i + z[i] > b + z[b]) b = i;
11    }
12 }

```

7.7. Manacher's Algorithm

```

1 int z[n];
2 void manacher(string s) {
3     // z[i] => longest odd palindrome centered at i is
4     //      s[i - z[i] ... i + z[i]]
5     // to get all palindromes (including even length),
6     // insert a '#' between each s[i] and s[i + 1]
7     int n = s.size();
8     z[0] = 0;
9     for (int b = 0, i = 1; i < n; i++) {
10        if (z[b] + b >= i)
11            z[i] = min(z[2 * b - i], b + z[b] - i);
12        else z[i] = 0;
13        while (i + z[i] + 1 < n && i - z[i] - 1 >= 0 &&
14              s[i + z[i] + 1] == s[i - z[i] - 1])
15            z[i]++;
16        if (z[i] + i > z[b] + b) b = i;
17    }
18 }

```

7.8. Minimum Rotation

```

1 int min_rotation(string s) {
2     int a = 0, n = s.size();
3     s += s;
4     for (int b = 0; b < n; b++) {
5         for (int k = 0; k < n; k++) {
6             if (a + k == b || s[a + k] < s[b + k]) {
7                 b += max(0, k - 1);
8             }
9         }
10    }
11 }

```

```

9         break;
10    }
11    if (s[a + k] > s[b + k]) {
12        a = b;
13        break;
14    }
15 }
16 return a;
17 }

```

7.9. Palindromic Tree

```

1
2 struct palindromic_tree {
3     struct node {
4         int next[26], fail, len;
5         int cnt,
6         num; // cnt: appear times, num: number of pal. suf.
7         node(int l = 0) : fail(0), len(l), cnt(0), num(0) {
8             for (int i = 0; i < 26; ++i) next[i] = 0;
9         }
10    };
11    vector<node> St;
12    vector<char> s;
13    int last, n;
14    palindromic_tree() : St(2), last(1), n(0) {
15        St[0].fail = 1, St[1].len = -1, s.pb(-1);
16    }
17    inline void clear() {
18        St.clear(), s.clear(), last = 1, n = 0;
19        St.pb(0), St.pb(-1);
20        St[0].fail = 1, s.pb(-1);
21    }
22    inline int get_fail(int x) {
23        while (s[n - St[x].len - 1] != s[n]) x = St[x].fail;
24        return x;
25    }
26    inline void add(int c) {
27        s.push_back(c -= 'a'), ++n;
28        int cur = get_fail(last);
29        if (!St[cur].next[c]) {
30            int now = SZ(St);
31            St.pb(St[cur].len + 2);
32            St[now].fail = St[get_fail(St[cur].fail)].next[c];
33            St[cur].next[c] = now;
34            St[now].num = St[St[now].fail].num + 1;
35        }
36        last = St[cur].next[c], ++St[last].cnt;
37    }
38    inline void count() { // counting cnt
39        auto i = St.rbegin();
40        for (; i != St.rend(); ++i) {
41            St[i->fail].cnt += i->cnt;
42        }
43    }
44    inline int size() { // The number of diff. pal.
45        return SZ(St) - 2;
46    }
47 }

```

8. Debug List

- 1 - Pre-submit:
 - Did you make a typo when copying a template?
 - Test more cases if unsure.
 - Write a naive solution and check small cases.
 - Submit the correct file.
- 7 - General Debugging:
 - Read the whole problem again.
 - Have a teammate read the problem.
 - Have a teammate read your code.
 - Explain your solution to them (or a rubber duck).
 - Print the code and its output / debug output.
 - Go to the toilet.
- 15 - Wrong Answer:
 - Any possible overflows?
 - > `__int128` ?
 - Try `#pragma GCC optimize("trapv")`
 - Floating point errors?
 - > `long double` ?
 - turn off math optimizations
 - check for `==`, `>=`, `acos(1.000000001)` , etc.
 - Did you forget to sort or unique?
 - Generate large and worst "corner" cases.
 - Check your `m` / `n`, `i` / `j` and `x` / `y`.
 - Are everything initialized or reset properly?
 - Are you sure about the STL thing you are using?
 - Read cppreference (should be available).

- ```
29 | - Print everything and run it on pen and paper.
31 | - Time Limit Exceeded:
32 | - Calculate your time complexity again.
33 | - Does the program actually end?
34 | - Check for `while(q.size())` etc.
35 | - Test the largest cases locally.
36 | - Did you do unnecessary stuff?
37 | - e.g. pass vectors by value
38 | - e.g. `memset` for every test case
39 | - Is your constant factor reasonable?
40 |
41 | - Runtime Error:
42 | - Check memory usage.
43 | - Forget to clear or destroy stuff?
44 | - > `vector::shrink_to_fit()`
45 | - Stack overflow?
46 | - Bad pointer / array access?
47 | - Try `fsanitize=address`
 - Division by zero? NaN's?
```