

Contents

1 Misc	1	4.2 Combinatorics	14
1.1 Contest	1	4.2.1 Matroid Intersection	14
1.1.1 Makefile	1	4.2.2 De Bruijn Sequence	15
1.2 How Did We Get Here?	2	4.2.3 Multinomial	15
1.2.1 The Basics	2	4.3 Algebra	15
1.2.2 Linux	2	4.3.1 Formal Power Series	15
1.2.3 Macros	2	4.4 Theorems	16
1.3 Tools	2	4.4.1 Kirchhoff's Theorem	16
1.3.1 Floating Point Binary Search	2	4.4.2 Tutte's Matrix	16
1.3.2 SplitMix64	2	4.4.3 Cayley's Formula	16
1.3.3 <random>	2	4.4.4 Erdős–Gallai Theorem	16
1.3.4 x86 Stack Hack	2	4.4.5 Burnside's Lemma	16
1.4 Algorithms	2	5 Randomisation	16
1.4.1 Bit Hacks	2	5.1 Simulated Annealing	16
1.4.2 Aliens Trick	2	6 Numeric	17
1.4.3 Hilbert Curve	2	6.1 Barrett Reduction	17
1.4.4 Infinite Grid Knight Distance	2	6.2 Long Long Multiplication	17
1.4.5 Poker Hand	2	6.3 Fast Fourier Transform	17
1.4.6 Longest Increasing Subsequence	3	6.4 Fast Walsh-Hadamard Transform	17
1.4.7 Mo's Algorithm on Tree	3	6.5 Subset Convolution	17
2 Data Structures	3	6.6 Linear Recurrences	17
2.1 GNU PBDS	3	6.6.1 Berlekamp-Massey Algorithm	17
2.2 Segment Tree (ZKW)	3	6.6.2 Linear Recurrence Calculation	17
2.3 Line Container	3	6.7 Matrices	18
2.4 Li-Chao Tree	3	6.7.1 Determinant	18
2.5 Heavy-Light Decomposition	4	6.7.2 Inverse	18
2.6 Wavelet Matrix	4	6.7.3 Characteristic Polynomial	18
2.7 Link-Cut Tree	5	6.7.4 Solve Linear Equation	19
3 Graph	5	6.8 Polynomial Interpolation	19
3.1 Floyd-Warshall	5	6.9 Simplex Algorithm	19
3.2 Dijkstra	5	7 Geometry	20
3.3 Modeling	6	7.1 Point	20
3.4 Matching/Flows	6	7.1.1 Quaternion	20
3.4.1 Dinic's Algorithm	6	7.1.2 Spherical Coordinates	20
3.4.2 Minimum Cost Flow	6	7.2 Segments	20
3.4.3 Gomory-Hu Tree	7	7.3 Convex Hull	20
3.4.4 Global Minimum Cut	7	7.3.1 3D Hull	21
3.4.5 Bipartite Minimum Cover	7	7.4 Angular Sort	21
3.4.6 Edmonds' Algorithm	7	7.5 Convex Polygon Minkowski Sum	21
3.4.7 Minimum Weight Matching	8	7.6 Point In Polygon	21
3.4.8 Stable Marriage	8	7.6.1 Convex Version	21
3.4.9 Kuhn-Munkres algorithm	9	7.6.2 Offline Multiple Points Version	22
3.5 Shortest Path Faster Algorithm	9	7.7 Closest Pair	22
3.6 Strongly Connected Components	9	7.8 Minimum Enclosing Circle	22
3.6.1 2-Satisfiability	10	7.9 Delaunay Triangulation	22
3.7 Biconnected Components	10	7.9.1 Slower Version	23
3.7.1 Articulation Points	10	7.10 Half Plane Intersection	23
3.7.2 Bridges	10	8 Strings	23
3.7.3 Block Cut Tree	10	8.1 Knuth-Morris-Pratt Algorithm	23
3.8 Triconnected Components	10	8.2 Aho-Corasick Automaton	24
3.9 Centroid Decomposition	11	8.3 Suffix Array	24
3.10 Minimum Mean Cycle	11	8.4 Suffix Tree	24
3.11 Directed MST	11	8.5 Z Value	25
3.12 Maximum Clique	12	8.6 Manacher's Algorithm	25
3.13 Dominator Tree	12	8.7 Minimum Rotation	25
3.14 Manhattan Distance MST	13	8.8 Palindromic Tree	25
4 Math	13	9 Debug List	25
4.1 Number Theory	13	1. Misc	
4.1.1 Mod Struct	13	1.1. Contest	
4.1.2 Miller-Rabin	13	1.1.1. Makefile	
4.1.3 Linear Sieve	13	1 .PRECIOUS: ./p%	
4.1.4 Get Factors	13	3 %: p%	
4.1.5 Binary GCD	13	5 ulimit -s unlimited && ./<	
4.1.6 Extended GCD	13	5 p%: p%.cpp	
4.1.7 Chinese Remainder Theorem	14	7 g++ -o \$@ \$< -std=c++17 -Wall -Wextra -Wshadow \	
4.1.8 Baby-Step Giant-Step	14	-fsanitize=address,undefined	
4.1.9 Pollard's Rho	14		
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		

1.2. How Did We Get Here?

1.2.1. The Basics

```
1 #include <bits/stdc++.h>
2 #define ll long long
3 #define sz(x) (int)(x).size()
4 using namespace std;
5 // mt19937
6 // rng(chrono::steady_clock::now().time_since_epoch().count());
7 // uniform_int_distribution<int>(1000,10000)(rng)

9 int main() {
10     ios_base::sync_with_stdio(0);
11     cin.tie(0);
12     return 0;
13 }
```

1.2.2. Linux

- g++ test.cpp
- ./test
- ./test < input.txt > output.txt (read/write to file)

1.2.3. Macros

Use vectorizations and math optimizations at your own peril.
For gcc≥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
2 #define _GLIBCXX_SANITIZE_VECTOR 1 // for asan on vectors
3 #pragma GCC optimize("O3", "unroll-loops")
4 #pragma GCC optimize("fast-math")
5 #pragma GCC target("avx,avx2,abm,bmi,bmi2") // tip: `lscpu`
6 // before a loop
7 #pragma GCC unroll 16 // 0 or 1 -> no unrolling
8 #pragma GCC ivdep
```

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))
12             r = m;
13         else
14             l = m;
15     }
16     return l.d;
17 }
```

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)
9             r = m;
10        else
11            l = m + 1;
12    }
13    return get_dp(l).first - l * k;
14 }
```

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;
12 }
```

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 using namespace std;
2
3 struct hand {
4     static constexpr auto rk = [] {
5         array<int, 256> x{};
6         auto s = "23456789TJQKACDHS";
7         for (int i = 0; i < 17; i++) x[s[i]] = i % 13;
8         return x;
9     }();
10    vector<pair<int, int>> v;
11    vector<int> cnt, vf, vs;
12    int type;
13    hand() : cnt(4), type(0) {}
14    void add_card(char suit, char rank) {
15        ++cnt[rk[suit]];
16        for (auto &[f, s] : v)
17            if (s == rk[rank]) return ++f, void();
18        v.emplace_back(1, rk[rank]);
19    }
20    void process() {
```

```

21 sort(v.rbegin(), v.rend());
22 for (auto [f, s] : v) vf.push_back(f), vs.push_back(s);
23 bool str = 0, flu = find(all(cnt), 5) != cnt.end();
24 if ((str = v.size() == 5))
25     for (int i = 1; i < 5; i++)
26         if (vs[i] != vs[i - 1] + 1) str = 0;
27 if (vs == vector<int>{12, 3, 2, 1, 0})
28     str = 1, vs = {3, 2, 1, 0, -1};
29 if (str && flu)
30     type = 9;
31 else if (vf[0] == 4)
32     type = 8;
33 else if (vf[0] == 3 && vf[1] == 2)
34     type = 7;
35 else if (str || flu)
36     type = 5 + flu;
37 else if (vf[0] == 3)
38     type = 4;
39 else if (vf[0] == 2)
40     type = 2 + (vf[1] == 2);
41 else
42     type = 1;
43 }
44 bool operator<(const hand &b) const {
45     return make_tuple(type, vf, vs) <
46            make_tuple(b.type, b.vf, b.vs);
47 }
};

```

1.4.6. Longest Increasing Subsequence

```

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

```

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)
14             l[i] = tin[u[i]], r[i] = tin[v[i]];
15         else
16             l[i] = tout[u[i]], r[i] = tin[v[i]];
17         qr[i] = i;
18     }
19     sort(qr.begin(), qr.end(), [&](int i, int j) {
20         if (l[i] / KB == l[j] / KB) return r[i] < r[j];
21         return l[i] / KB < l[j] / KB;
22     });
23     vector<bool> used(n);
24     // Add(v): add/remove v to/from the path based on used[v]
25     for (int i = 0, tl = 0, tr = -1; i < q; ++i) {
26         while (tl < l[qr[i]]) Add(euler[tl++]);
27         while (tl > l[qr[i]]) Add(euler[tl--]);
28         while (tr > r[qr[i]]) Add(euler[tr--]);
29         while (tr < r[qr[i]]) Add(euler[tr+1]);
30         // add/remove LCA(u, v) if necessary
31     }
32 }

```

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 // most std::map + order_of_key, find_by_order, split, join
6 template<typename T, typename U = null_type>
7 using ordered_map = tree<T, U, std::less<>, rb_tree_tag,
8                        tree_order_statistics_node_update>;
9 // useful tags: rb_tree_tag, splay_tree_tag

10 template<typename T> struct myhash {
11     size_t operator()(T x) const; // splitmix, bswap(x*R), ...
12 };
13 // most of std::unordered_map, but faster (needs good hash)
14 template<typename T, typename U = null_type>
15 using hash_table = gp_hash_table<T, U, myhash<T>>;

16 // most std::priority_queue + modify, erase, split, join
17 using heap = priority_queue<int, std::less<>>;
18 // useful tags: pairing_heap_tag, binary_heap_tag,
19 //              (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 struct Line {
2     mutable ll k, m, p;
3     bool operator<(const Line &o) const { return k < o.k; }
4     bool operator<(ll x) const { return p < x; }
5 };
6 // add: line y=kx+m, query: maximum y of given x
7 struct LineContainer : multiset<Line, less<>> {
8     // (for doubles, use inf = 1/.0, div(a,b) = a/b)
9     static const ll inf = LLONG_MAX;
10    ll div(ll a, ll b) { // floored division
11        return a / b - ((a ^ b) < 0 && a % b);
12    }
13    bool isect(iterator x, iterator y) {
14        if (y == end()) return x->p = inf, 0;
15        if (x->k == y->k)
16            x->p = x->m > y->m ? inf : -inf;
17        else
18            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    }
34 };

```

2.4. Li-Chao Tree

```

1 constexpr ll MAXN = 2e5, INF = 2e18;
2 struct Line {

```

```

3  ll m, b;
   Line() : m(0), b(-INF) {}
5  Line(ll _m, ll _b) : m(_m), b(_b) {}
   ll operator()(ll x) const { return m * x + b; }
7  };
   struct Li_Chao {
9     Line a[MAXN * 4];
   void insert(Line seg, int l, int r, int v = 1) {
11         if (l == r) {
           if (seg(l) > a[v](l)) a[v] = seg;
13         return;
       }
15         int mid = (l + r) >> 1;
       if (a[v].m > seg.m) swap(a[v], seg);
17         if (a[v](mid) < seg(mid)) {
           swap(a[v], seg);
           insert(seg, l, mid, v << 1);
21         } else {
           insert(seg, mid + 1, r, v << 1 | 1);
       }
23     }
   ll query(int x, int l, int r, int v = 1) {
       if (l == r) return a[v](x);
25         int mid = (l + r) >> 1;
       if (x <= mid)
27             return max(a[v](x), query(x, l, mid, v << 1));
       else
29             return max(a[v](x), query(x, mid + 1, r, v << 1 | 1));
31     };

```

2.5. Heavy-Light Decomposition

```

1  struct heavy_light_decomposition {
   int n;
   vector<vector<int>> edges;
   vector<int> par, heavy, height, pos, head;
5  heavy_light_decomposition(int n)
       : n(n), edges(n + 1), par(n + 1), heavy(n + 1),
         height(n + 1), pos(n + 1), head(n + 1) {}
7  void add_edge(int x, int y) {
   edges[x].push_back(y);
   edges[y].push_back(x);
11 }
   void heavy_dfs(int c, vector<int> &sub) {
13     heavy[c] = -1;
   sub[c] = 1;
15     int max_size = 0;
   for (int i : edges[c]) {
17         if (i != par[c]) {
           par[i] = c;
           height[i] = height[c] + 1;
           heavy_dfs(i, sub);
           sub[c] += sub[i];
21         if (sub[i] > max_size) {
           heavy[c] = i;
           max_size = sub[i];
23         }
       }
25     }
   }
27 }
   void decompose(int c, int h, int &timer) {
31     pos[c] = timer++;
   head[c] = h;
   if (heavy[c] != -1) { decompose(heavy[c], h, timer); }
33     for (int i : edges[c]) {
       if (i != par[c] && i != heavy[c]) {
35         decompose(i, i, timer);
       }
37     }
   }
39     void build() {
       int timer = 0;
       vector<int> sub(n + 1);
       for (int i = 1; i <= n; ++i) {
43         if (sub[i] == 0) {
           par[i] = 0;
           height[i] = 0;
           heavy_dfs(i, sub);
           decompose(i, i, timer);
47         }
       }
49     }
   int get_position(int x) { return pos[x]; }
   vector<array<int, 2>> path_queries(int x, int y) {
53     vector<array<int, 2>> queries;
   while (head[x] != head[y]) {
55         if (height[head[x]] > height[head[y]]) { swap(x, y); }
       queries.push_back({pos[head[y]], pos[y]});
57         y = par[head[y]];
   }

```

```

59     if (height[x] > height[y]) { swap(x, y); }
   queries.push_back({pos[x], pos[y]});
61     return queries;
   }
63 };

```

2.6. Wavelet Matrix

```

1  #pragma GCC target("popcnt,bmi2")
   #include <immintrin.h>
3
   // T is unsigned. You might want to compress values first
5  template <typename T> struct wavelet_matrix {
   static_assert(is_unsigned_v<T>, "only unsigned T");
7     struct bit_vector {
       static constexpr uint W = 64;
       uint n, cnt0;
       vector<ull> bits;
       vector<uint> sum;
       bit_vector(uint n_)
11           : n(n_), bits(n / W + 1), sum(n / W + 1) {}
       void build() {
13           for (uint j = 0; j != n / W; ++j)
               sum[j + 1] = sum[j] + _mm_popcnt_u64(bits[j]);
15           cnt0 = rank0(n);
       }
17         void set_bit(uint i) { bits[i / W] |= 1ULL << i % W; }
       bool operator[](uint i) const {
21             return !(bits[i / W] & 1ULL << i % W);
       }
23         uint rank1(uint i) const {
           return sum[i / W] +
25             _mm_popcnt_u64(_bzhi_u64(bits[i / W], i % W));
       }
27         uint rank0(uint i) const { return i - rank1(i); }
   };
   uint n, lg;
   vector<bit_vector> b;
   wavelet_matrix(const vector<T> &a) : n(a.size()) {
31       lg =
           __lg(max(*max_element(a.begin(), a.end()), T(1))) + 1;
       b.assign(lg, n);
33       vector<T> cur = a, nxt(n);
       for (int h = lg; h--;) {
35           for (uint i = 0; i < n; ++i)
               if (cur[i] & (T(1) << h)) b[h].set_bit(i);
37           b[h].build();
           int il = 0, ir = b[h].cnt0;
39           for (uint i = 0; i < n; ++i)
               nxt[(b[h][i] ? ir : il)++] = cur[i];
41           swap(cur, nxt);
43       }
45     }
   T operator[](uint i) const {
47         T res = 0;
       for (int h = lg; h--;)
49             if (b[h][i])
                 i += b[h].cnt0 - b[h].rank0(i), res |= T(1) << h;
51         return res;
53     }
   // query k-th smallest (0-based) in a[l, r)
55     T kth(uint l, uint r, uint k) const {
       T res = 0;
57         for (int h = lg; h--;) {
           uint tl = b[h].rank0(l), tr = b[h].rank0(r);
59           if (k >= tr - tl) {
               k -= tr - tl;
               l += b[h].cnt0 - tl;
               r += b[h].cnt0 - tr;
61               res |= T(1) << h;
           } else {
               l = tl, r = tr;
63           }
       }
65         return res;
67     }
   // count of i in [l, r) with a[i] < u
69     uint count(uint l, uint r, T u) const {
       if (u >= T(1) << lg) return r - l;
71         uint res = 0;
       for (int h = lg; h--;) {
73             uint tl = b[h].rank0(l), tr = b[h].rank0(r);
           if (u & (T(1) << h)) {
75                 l += b[h].cnt0 - tl;
                 r += b[h].cnt0 - tr;
77                 res += tr - tl;
           } else {
79                 l = tl, r = tr;
81             }
       }
   }

```

```

83     return res;
84 }
85 };

```

2.7. Link-Cut Tree

```

1  const int MXN = 100005;
2  const int MEM = 100005;
3
4  struct Splay {
5      static Splay nil, mem[MEM], *pmem;
6      Splay *ch[2], *f;
7      int val, rev, size;
8      Splay() : val(-1), rev(0), size(0) {
9          f = ch[0] = ch[1] = &nil;
10     }
11     Splay(int _val) : val(_val), rev(0), size(1) {
12         f = ch[0] = ch[1] = &nil;
13     }
14     bool isr() {
15         return f->ch[0] != this && f->ch[1] != this;
16     }
17     int dir() { return f->ch[0] == this ? 0 : 1; }
18     void setCh(Splay *c, int d) {
19         ch[d] = c;
20         if (c != &nil) c->f = this;
21         pull();
22     }
23     void push() {
24         if (rev) {
25             swap(ch[0], ch[1]);
26             if (ch[0] != &nil) ch[0]->rev ^= 1;
27             if (ch[1] != &nil) ch[1]->rev ^= 1;
28             rev = 0;
29         }
30     }
31     void pull() {
32         size = ch[0]->size + ch[1]->size + 1;
33         if (ch[0] != &nil) ch[0]->f = this;
34         if (ch[1] != &nil) ch[1]->f = this;
35     }
36 } Splay::nil, Splay::mem[MEM], *Splay::pmem = Splay::mem;
37 Splay *nil = &Splay::nil;
38
39 void rotate(Splay *x) {
40     Splay *p = x->f;
41     int d = x->dir();
42     if (!p->isr())
43         p->f->setCh(x, p->dir());
44     else
45         x->f = p->f;
46     p->setCh(x->ch[!d], d);
47     x->setCh(p, !d);
48     p->pull();
49     x->pull();
50 }
51
52 vector<Splay *> splayVec;
53 void splay(Splay *x) {
54     splayVec.clear();
55     for (Splay *q = x;; q = q->f) {
56         splayVec.push_back(q);
57         if (q->isr()) break;
58     }
59     reverse(begin(splayVec), end(splayVec));
60     for (auto it : splayVec) it->push();
61     while (!x->isr()) {
62         if (x->f->isr())
63             rotate(x);
64         else if (x->dir() == x->f->dir())
65             rotate(x->f), rotate(x);
66         else
67             rotate(x), rotate(x);
68     }
69 }
70
71 Splay *access(Splay *x) {
72     Splay *q = nil;
73     for (; x != nil; x = x->f) {
74         splay(x);
75         x->setCh(q, 1);
76         q = x;
77     }
78     return q;
79 }
80
81 void evert(Splay *x) {
82     access(x);
83     splay(x);
84     x->rev ^= 1;
85     x->push();

```

```

85     x->pull();
86 }
87 void link(Splay *x, Splay *y) {
88     // evert(x);
89     access(x);
90     splay(x);
91     evert(y);
92     x->setCh(y, 1);
93 }
94
95 void cut(Splay *x, Splay *y) {
96     // evert(x);
97     access(y);
98     splay(y);
99     y->push();
100    y->ch[0] = y->ch[0]->f = nil;
101 }
102
103 int N, Q;
104 Splay *vt[MXN];
105
106 int ask(Splay *x, Splay *y) {
107     access(x);
108     access(y);
109     splay(x);
110     int res = x->f->val;
111     if (res == -1) res = x->val;
112     return res;
113 }
114
115 int main(int argc, char **argv) {
116     scanf("%d%d", &N, &Q);
117     for (int i = 1; i <= N; i++)
118         vt[i] = new (Splay::pmem++) Splay(i);
119     while (Q--) {
120         char cmd[105];
121         int u, v;
122         scanf("%s", cmd);
123         if (cmd[1] == 'i') {
124             scanf("%d%d", &u, &v);
125             link(vt[u], vt[v]);
126         } else if (cmd[0] == 'c') {
127             scanf("%d", &v);
128             cut(vt[1], vt[v]);
129         } else {
130             scanf("%d%d", &u, &v);
131             int res = ask(vt[u], vt[v]);
132             printf("%d\n", res);
133         }
134     }
135 }

```

3. Graph

3.1. Floyd-Warshall

```

1  for (int k = 0; k < n; ++k) {
2      for (int i = 0; i < n; ++i) {
3          for (int j = 0; j < n; ++j) {
4              d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
5          }
6      }
7  }

```

3.2. Dijkstra

```

1  struct dijkstra {
2      using S =
3      int; // regular edge weight type, change if needed
4      using T = long long; // maximum possible distance type,
5      // change if needed
6
7      const T INF = 1e18;
8      struct Edge {
9          int target;
10         S weight;
11     };
12     struct next_shortest {
13         int node;
14         T dist;
15         bool operator<(const next_shortest &rhs) const {
16             return dist > rhs.dist;
17         }
18     };
19     int n;
20     vector<vector<Edge>> e;
21     dijkstra(int n) : n(n), e(n + 1) {}
22     void add_directed_edge(int x, int y, S z) {
23         e[x].push_back({y, z});
24     }
25     void add_undirected_edge(int x, int y, S z) {

```



```

25     e[x].push_back({y, z});
26     e[y].push_back({x, z});
27 }
28 vector<T> shortest_path(int src) {
29     vector<T> d(n + 1, INF);
30     d[src] = 0;
31     priority_queue<next_shortest> pq;
32     pq.push({src, 0});
33     while (!pq.empty()) {
34         int cur_node = pq.top().node;
35         T cur_dist = pq.top().dist;
36         pq.pop();
37         if (cur_dist > d[cur_node]) continue;
38         for (auto i : e[cur_node]) {
39             if (cur_dist + i.weight < d[i.target]) {
40                 d[i.target] = cur_dist + i.weight;
41                 pq.push({i.target, d[i.target]});
42             }
43         }
44     }
45     return d;
46 };

```

3.3. Modeling

- Maximum/Minimum flow with lower bound / Circulation problem
 - Construct super source S and sink T .
 - For each edge (x, y, l, u) , connect $x \rightarrow y$ with capacity $u - l$.
 - For each vertex v , denote by $in(v)$ the difference between the sum of incoming lower bounds and the sum of outgoing lower bounds.
 - 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.
 - 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)
 - Redirect every edge: $y \rightarrow x$ if $(x, y) \in M$, $x \rightarrow y$ otherwise.
 - DFS from unmatched vertices in X .
 - $x \in X$ is chosen iff x is unvisited.
 - $y \in Y$ is chosen iff y is visited.
- Minimum cost cyclic flow
 - Construct super source S and sink T
 - For each edge (x, y, c) , connect $x \rightarrow y$ with $(cost, cap) = (c, 1)$ if $c > 0$, otherwise connect $y \rightarrow x$ with $(cost, cap) = (-c, 1)$
 - For each edge with $c < 0$, sum these cost as K , then increase $d(y)$ by 1, decrease $d(x)$ by 1
 - For each vertex v with $d(v) > 0$, connect $S \rightarrow v$ with $(cost, cap) = (0, d(v))$
 - For each vertex v with $d(v) < 0$, connect $v \rightarrow T$ with $(cost, cap) = (0, -d(v))$
 - Flow from S to T , the answer is the cost of the flow $C + K$
- Maximum density induced subgraph
 - Binary search on answer, suppose we're checking answer T
 - Construct a max flow model, let K be the sum of all weights
 - Connect source $s \rightarrow v$, $v \in G$ with capacity K
 - For each edge (u, v, w) in G , connect $u \rightarrow v$ and $v \rightarrow u$ with capacity w
 - For $v \in G$, connect it with sink $v \rightarrow t$ with capacity $K + 2T - (\sum_{e \in E(v)} w(e)) - 2w(v)$
 - T is a valid answer if the maximum flow $f < K|V|$
- Minimum weight edge cover
 - For each $v \in V$ create a copy v' , and connect $u' \rightarrow v'$ with weight $w(u, v)$.
 - Connect $v \rightarrow v'$ with weight $2\mu(v)$, where $\mu(v)$ is the cost of the cheapest edge incident to v .
 - Find the minimum weight perfect matching on G' .
- Project selection problem
 - If $p_v > 0$, create edge (s, v) with capacity p_v ; otherwise, create edge (v, t) with capacity $-p_v$.
 - Create edge (u, v) with capacity w with w being the cost of choosing u without choosing v .
 - 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:

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

3.4. Matching/Flows

3.4.1. 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        return flow;
48    }
49    void reset() {
50        fill_n(side, MAXN, 0);
51        for (auto &i : v) i.clear();
52    }
53 };

```

3.4.2. 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});
18        while (!q.empty()) {
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});

```

```

    else
        q.modify(its[e.to], {-dis[e.to], e.to});
    }
}
}
}
bool AP(ll &flow) {
    fill_n(dis, n, INF);
    fromE[s] = 0;
    dis[s] = 0;
    flows[s] = flowlim - flow;
    dijkstra();
    if (dis[t] == INF) return false;
    flow += flows[t];
    for (edge *e = fromE[t]; e; e = fromE[e->from]) {
        e->flow += flows[t];
        v[e->to][e->rev].flow -= flows[t];
    }
    for (int i = 0; i < n; i++)
        pi[i] = min(pi[i] + dis[i], INF);
    return true;
}
pll solve(int _s, int _t, ll _flowlim = INF) {
    s = _s, t = _t, flowlim = _flowlim;
    pll re;
    while (re.F != flowlim && AP(re.F))
        ;
    for (int i = 0; i < n; i++)
        for (edge &e : v[i])
            if (e.flow != 0) re.S += e.flow * e.cost;
    re.S /= 2;
    return re;
}
void init(int _n) {
    n = _n;
    fill_n(pi, n, 0);
    for (int i = 0; i < n; i++) v[i].clear();
}
void setpi(int s) {
    fill_n(pi, n, INF);
    pi[s] = 0;
    for (ll it = 0, flag = 1, tdis; flag && it < n; it++) {
        flag = 0;
        for (int i = 0; i < n; i++)
            if (pi[i] != INF)
                for (edge &e : v[i])
                    if (e.cap && (tdis = pi[i] + e.cost) < pi[e.to])
                        pi[e.to] = tdis, flag = 1;
    }
}
};

```

3.4.3. Gomory-Hu Tree

Requires: Dinic's Algorithm

```

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

```

3.4.4. Global Minimum Cut

```

1 // weights is an adjacency matrix, undirected
2 pair<int, vi> getMinCut(vector<vi> &weights) {
3     int N = sz(weights);
4     vi used(N), cut, best_cut;
5     int best_weight = -1;
6
7     for (int phase = N - 1; phase >= 0; phase--) {
8         vi w = weights[0], added = used;
9         int prev, k = 0;
10        rep(i, 0, phase) {
11            prev = k;
12            k = -1;
13            rep(j, 1, N) if (!added[j] &&
14                            (k == -1 || w[j] > w[k])) k = j;
15            if (i == phase - 1) {
16                rep(j, 0, N) weights[prev][j] += weights[k][j];

```

```

17            rep(j, 0, N) weights[j][prev] = weights[prev][j];
18            used[k] = true;
19            cut.push_back(k);
20            if (best_weight == -1 || w[k] < best_weight) {
21                best_cut = cut;
22                best_weight = w[k];
23            }
24        } else {
25            rep(j, 0, N) w[j] += weights[k][j];
26            added[k] = true;
27        }
28    }
29    return {best_weight, best_cut};
30 }

```

3.4.5. Bipartite Minimum Cover

Requires: Dinic's Algorithm

```

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

```

3.4.6. Edmonds' Algorithm

```

1 struct Edmonds {
2     int n, T;
3     vector<vector<int>> g;
4     vector<int> pa, p, used, base;
5     Edmonds(int n)
6         : n(n), T(0), g(n), pa(n, -1), p(n), used(n),
7           base(n) {}
8     void add(int a, int b) {
9         g[a].push_back(b);
10        g[b].push_back(a);
11    }
12    int getBase(int i) {
13        while (i != base[i])
14            base[i] = base[base[i]], i = base[i];
15        return i;
16    }
17    vector<int> toJoin;
18    void mark_path(int v, int x, int b, vector<int> &path) {
19        for (; getBase(v) != b; v = p[x]) {
20            p[v] = x, x = pa[v];

```

```

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

```

3.4.7. 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 (u != v && match[u] != v && !onstk[v]) {
23                 int m = match[v];
24                 if (d[m] > d[u] - e[v][m] + e[u][v]) {
25                     d[m] = d[u] - e[v][m] + e[u][v];
26                     onstk[v] = 1;
27                     stk.push_back(v);

```

```

29             if (SPFA(m)) return true;
30             stk.pop_back();
31             onstk[v] = 0;
32         }
33     }
34     onstk[u] = 0;
35     stk.pop_back();
36     return false;
37 }
38 int solve() {
39     for (int i = 0; i < n; i += 2) {
40         match[i] = i + 1;
41         match[i + 1] = i;
42     }
43     while (true) {
44         int found = 0;
45         for (int i = 0; i < n; i++) onstk[i] = d[i] = 0;
46         for (int i = 0; i < n; i++) {
47             stk.clear();
48             if (!onstk[i] && SPFA(i)) {
49                 found = 1;
50                 while (stk.size() >= 2) {
51                     int u = stk.back();
52                     stk.pop_back();
53                     int v = stk.back();
54                     stk.pop_back();
55                     match[u] = v;
56                     match[v] = u;
57                 }
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 } graph;

```

3.4.8. Stable Marriage

```

1 // normal stable marriage problem
2 /* input:
3 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
10 */
11
12 using namespace std;
13 const int MAXN = 505;
14
15 int n;
16 int favor[MAXN][MAXN]; // favor[boy_id][rank] = girl_id;
17 int order[MAXN][MAXN]; // order[girl_id][boy_id] = rank;
18 int current[MAXN]; // current[boy_id] = rank;
19 // boy_id will pursue current[boy_id] girl.
20 int girl_current[MAXN]; // girl[girl_id] = boy_id;
21
22 void initialize() {
23     for (int i = 0; i < n; i++) {
24         current[i] = 0;
25         girl_current[i] = n;
26         order[i][n] = n;
27     }
28 }
29
30 map<string, int> male, female;
31 string bname[MAXN], gname[MAXN];
32 int fit = 0;
33
34 void stable_marriage() {
35     queue<int> que;
36     for (int i = 0; i < n; i++) que.push(i);
37     while (!que.empty()) {
38         int boy_id = que.front();
39         que.pop();
40
41         int girl_id = favor[boy_id][current[boy_id]];
42         current[boy_id]++;
43
44         if (order[girl_id][boy_id] <
45             order[girl_id][girl_current[girl_id]]) {
46             if (girl_current[girl_id] < n)

```



```

49     que.push(girl_current[girl_id]);
51     girl_current[girl_id] = boy_id;
52 } else {
53     que.push(boy_id);
54 }
55 }
57 int main() {
58     cin >> n;
59
60     for (int i = 0; i < n; i++) {
61         string p, t;
62         cin >> p;
63         male[p] = i;
64         bname[i] = p;
65         for (int j = 0; j < n; j++) {
66             cin >> t;
67             if (!female.count(t)) {
68                 gname[fit] = t;
69                 female[t] = fit++;
70             }
71             favor[i][j] = female[t];
72         }
73     }
74
75     for (int i = 0; i < n; i++) {
76         string p, t;
77         cin >> p;
78         for (int j = 0; j < n; j++) {
79             cin >> t;
80             order[female[p]][male[t]] = j;
81         }
82     }
83
84     initialize();
85     stable_marriage();
86
87     for (int i = 0; i < n; i++) {
88         cout << bname[i] << " "
89              << gname[favor[i][current[i] - 1]] << endl;
90     }
91 }

```

3.4.9. Kuhn-Munkres algorithm

```

1 // Maximum Weight Perfect Bipartite Matching
2 // Detect non-perfect-matching:
3 // 1. set all edge[i][j] as INF
4 // 2. if solve() >= INF, it is not perfect matching.
5
6 typedef long long ll;
7 struct KM {
8     static const int MAXN = 1050;
9     static const ll INF = 1LL << 60;
10    int n, match[MAXN], vx[MAXN], vy[MAXN];
11    ll edge[MAXN][MAXN], lx[MAXN], ly[MAXN], slack[MAXN];
12    void init(int n) {
13        n = n;
14        for (int i = 0; i < n; i++)
15            for (int j = 0; j < n; j++) edge[i][j] = 0;
16    }
17    void add_edge(int x, int y, ll w) { edge[x][y] = w; }
18    bool DFS(int x) {
19        vx[x] = 1;
20        for (int y = 0; y < n; y++) {
21            if (vy[y]) continue;
22            if (lx[x] + ly[y] > edge[x][y]) {
23                slack[y] = min(slack[y], lx[x] + ly[y] - edge[x][y]);
24            } else {
25                vy[y] = 1;
26                if (match[y] == -1 || DFS(match[y])) {
27                    match[y] = x;
28                    return true;
29                }
30            }
31        }
32        return false;
33    }
34    ll solve() {
35        fill(match, match + n, -1);
36        fill(lx, lx + n, -INF);
37        fill(ly, ly + n, 0);
38        for (int i = 0; i < n; i++)
39            for (int j = 0; j < n; j++)
40                lx[i] = max(lx[i], edge[i][j]);
41        for (int i = 0; i < n; i++) {
42            fill(slack, slack + n, INF);
43            while (true) {
44                fill(vx, vx + n, 0);

```

```

47         fill(vy, vy + n, 0);
48         if (DFS(i)) break;
49         ll d = INF;
50         for (int j = 0; j < n; j++)
51             if (!vy[j]) d = min(d, slack[j]);
52         for (int j = 0; j < n; j++) {
53             if (vx[j]) lx[j] -= d;
54             if (vy[j]) ly[j] += d;
55             else
56                 slack[j] -= d;
57         }
58     }
59     ll res = 0;
60     for (int i = 0; i < n; i++) {
61         res += edge[match[i]][i];
62     }
63     return res;
64 }
65 } graph;

```

3.5. 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            }
30            q.swap(tq);
31        }
32        bool re = !q.empty();
33        inneg.reset();
34        while (!q.empty()) {
35            if (!inneg[q.front()]) dfs(q.front());
36            q.pop();
37        }
38        return re;
39    }
40    void reset(int n) {
41        for (int i = 0; i <= n; i++) v[i].clear();
42    }
43 };

```

3.6. Strongly Connected Components

```

1 struct strongly_connected_components {
2     int n, component_count;
3     vector<vector<int>> edges, reverse_edges;
4     vector<int> component, component_size;
5     strongly_connected_components(int n)
6         : n(n), edges(n + 1), reverse_edges(n + 1),
7           component(n + 1), component_size(n + 1),
8           component_count(0) {}
9     void add_edge(int x, int y) {
10         edges[x].push_back(y);
11         reverse_edges[y].push_back(x);
12     }
13     void dfs(int c, vector<int> &out_order,
14              vector<bool> &done) {
15         done[c] = true;
16         for (int i : edges[c])
17             if (!done[i]) dfs(i, out_order, done);
18         out_order.push_back(c);
19     }
20     void dfs_reverse(int c, vector<bool> &done) {

```

```

21 done[c] = true;
22 component[c] = component_count;
23 ++component_size[component_count];
24 for (int i : reverse_edges[c])
25     if (!done[i]) dfs_reverse(i, done);
26 }
27 void build_scc() {
28     vector<int> out_order;
29     vector<bool> done(n + 1, false),
30     done_reverse(n + 1, false);
31     for (int i = 1; i <= n; ++i)
32         if (!done[i]) dfs(i, out_order, done);
33     component_count = 0;
34     for (int i = out_order.size() - 1; i >= 0; --i)
35         if (!done_reverse[out_order[i]]) {
36             ++component_count;
37             component_size[component_count] = 0;
38             dfs_reverse(out_order[i], done_reverse);
39         }
40 }
41 };

```

3.6.1. 2-Satisfiability

Requires: Strongly Connected Components

```

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

```

3.7. Biconnected Components

3.7.1. 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        }
26        if (ch == 1 && p == -1) cut[x] = false;
27 }

```

3.7.2. 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            if (u == x) break;
22        }
23    }
24 }

```

3.7.3. Block Cut Tree

```

1 struct block_cut_tree {
2     int n, edge_id, bct_node_cnt;
3     vector<vector<array<int, 2>>> edges;
4     vector<vector<int>> edges_bct;
5     vector<bool> is_split;
6     block_cut_tree(int n)
7         : n(n), edges(n + 1), edges_bct(n * 3 + 1),
8         edge_id(0) {}
9     void add_edge(int x, int y) {
10        edges[x].push_back({y, edge_id});
11        edges[y].push_back({x, edge_id});
12        ++edge_id;
13        is_split.push_back(false);
14    }
15    void dfs_split(int c, int pid, int &timer,
16        vector<int> &tin, vector<int> &low,
17        vector<bool> &done) {
18        done[c] = true;
19        tin[c] = low[c] = ++timer;
20        for (auto i : edges[c])
21            if (i[1] != pid) {
22                if (done[i[0]])
23                    low[c] = min(low[c], tin[i[0]]);
24                else {
25                    dfs_split(i[0], i[1], timer, tin, low, done);
26                    low[c] = min(low[c], low[i[0]]);
27                    is_split[i[1]] = tin[c] <= low[i[0]];
28                }
29            }
30    }
31    void dfs_build_bct(int c, int g, int &timer,
32        vector<bool> &done) {
33        done[c] = true;
34        edges_bct[c].push_back(g + n);
35        edges_bct[g + n].push_back(c);
36        for (auto i : edges[c])
37            if (!done[i[0]]) {
38                if (is_split[i[1]]) {
39                    ++timer;
40                    edges_bct[c].push_back(timer + n);
41                    edges_bct[timer + n].push_back(c);
42                    dfs_build_bct(i[0], timer, timer, done);
43                } else {
44                    dfs_build_bct(i[0], g, timer, done);
45                }
46            }
47    }
48    void build_bct() {
49        int timer = 0;
50        vector<int> tin(n + 1), low(n + 1);
51        vector<bool> done_dfs(n + 1), done_build(n + 1);
52        bct_node_cnt = 0;
53        for (int i = 1; i <= n; ++i)
54            if (!done_dfs[i]) {
55                dfs_split(i, -1, timer, tin, low, done_dfs);
56                ++bct_node_cnt;
57                dfs_build_bct(i, bct_node_cnt, bct_node_cnt,
58                    done_build);
59            }
60    }
61 };

```

3.8. Triconnected Components

```

1 // requires a union-find data structure
2 struct ThreeEdgeCC {
3     int V, ind;
4     vector<int> id, pre, post, low, deg, path;
5     vector<vector<int>> components;
6     UnionFind uf;
7     template <class Graph>
8     void dfs(const Graph &G, int v, int prev) {
9         pre[v] = ++ind;

```

```

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

```

3.9. 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        }
12    }
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     }
38 }

```

3.10. Minimum Mean Cycle

```

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

```

3.11. 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]) {
32                    int j = i, c = 0;
33                    while (j != root && fr[j] != i && c <= n)
34                        ++c, j = fr[j];
35                    if (j == root || c > n)
36                        continue;
37                    else {
38                        x = i;
39                        break;
40                    }
41                }
42            if (!x) break;
43            for (int i = 1; i <= n; ++i)
44                if (i != root && !inc[i]) ans += fw[i];
45            return ans;
46        }
47    }

```

```

47     int y = x;
49     for (int i = 1; i <= n; ++i) vis[i] = false;
51     do {
53         ans += fw[y];
55         y = fr[y];
57         vis[y] = inc[y] = true;
59         while (y != x);
61         inc[x] = false;
63         for (int k = 1; k <= n; ++k)
65             if (vis[k]) {
67                 for (int j = 1; j <= n; ++j)
69                     if (!vis[j]) {
71                         if (g[x][j] > g[k][j]) g[x][j] = g[k][j];
73                         if (g[j][k] < inf &&
75                             g[j][k] - fw[k] < g[j][x])
76                             g[j][x] = g[j][k] - fw[k];
77                     }
78             }
79     }
80     return ans;
81 }
82 int dfs(int now) {
83     int r = 1;
84     vis[now] = true;
85     for (int i = 1; i <= n; ++i)
86         if (g[now][i] < inf && !vis[i]) r += dfs(i);
87     return r;
88 }
89 }

```

3.12. Maximum Clique

```

1 // source: KACTL
3 typedef vector<bitset<200>> vb;
4 struct Maxclique {
5     double limit = 0.025, pk = 0;
6     struct Vertex {
7         int i, d = 0;
8     };
9     typedef vector<Vertex> vv;
10    vb e;
11    vv V;
12    vector<vi> C;
13    vi qmax, q, S, old;
14    void init(vv &r) {
15        for (auto &v : r) v.d = 0;
16        for (auto &v : r)
17            for (auto j : r) v.d += e[v.i][j.i];
18        sort(all(r), [](auto a, auto b) { return a.d > b.d; });
19        int mxD = r[0].d;
20        rep(i, 0, sz(r)) r[i].d = min(i, mxD) + 1;
21    }
22    void expand(vv &r, int lev = 1) {
23        S[lev] += S[lev - 1] - old[lev];
24        old[lev] = S[lev - 1];
25        while (sz(R)) {
26            if (sz(q) + R.back().d <= sz(qmax)) return;
27            q.push_back(R.back().i);
28            vv T;
29            for (auto v : R)
30                if (e[R.back().i][v.i]) T.push_back({v.i});
31            if (sz(T)) {
32                if (S[lev]++ / ++pk < limit) init(T);
33                int j = 0, mxk = 1,
34                    mnk = max(sz(qmax) - sz(q) + 1, 1);
35                C[1].clear(), C[2].clear();
36                for (auto v : T) {
37                    int k = 1;
38                    auto f = [&](int i) { return e[v.i][i]; };
39                    while (any_of(all(C[k]), f)) k++;
40                    if (k > mxk) mxk = k, C[mxk + 1].clear();
41                    if (k < mnk) T[j++].i = v.i;
42                    C[k].push_back(v.i);
43                }
44                if (j > 0) T[j - 1].d = 0;
45                rep(k, mnk, mxk + 1) for (int i : C[k]) T[j].i = i,
46                    T[j++].d = k;
47            }
48            expand(T, lev + 1);
49        } else if (sz(q) > sz(qmax))
50            qmax = q;
51        q.pop_back(), R.pop_back();
52    }
53 }
54 vi maxClique() {
55     init(V), expand(V);
56     return qmax;
57 }
58 Maxclique(vb conn)

```

```

59     : e(conn), C(sz(e) + 1), S(sz(C)), old(S) {
61         rep(i, 0, sz(e)) V.push_back({i});
62     };
63 }

```

3.13. 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]))
70                    idom[w] = mn[w];
71                else
72                    idom[w] = par[u];
73            }
74            cov[par[u]].clear();
75        }
76        REP1(i, 2, ts) {
77            int u = nfd[i];
78            if (u == 0) continue;
79            if (idom[u] != sdom[u]) idom[u] = idom[idom[u]];
80        }
81    } dom;

```

3.1.4. Manhattan Distance MST

```

1 // returns [(dist, from, to), ...]
  // then do normal mst afterwards
2 typedef Point<int> P;
3 vector<array<int, 3>> manhattanMST(vector<P> ps) {
4     vi id(sz(ps));
5     iota(all(id), 0);
6     vector<array<int, 3>> edges;
7     rep(k, 0, 4) {
8         sort(all(id), [&](int i, int j) {
9             return (ps[i] - ps[j]).x < (ps[j] - ps[i]).y;
10        });
11        map<int, int> sweep;
12        for (int i : id) {
13            for (auto it = sweep.lower_bound(-ps[i].y);
14                 it != sweep.end(); sweep.erase(it++)) {
15                int j = it->second;
16                P d = ps[i] - ps[j];
17                if (d.y > d.x) break;
18                edges.push_back({d.y + d.x, i, j});
19            }
20            sweep[-ps[i].y] = i;
21        }
22        for (P &p : ps)
23            if (k & 1)
24                p.x = -p.x;
25            else
26                swap(p.x, p.y);
27        }
28    }
29    return edges;
30 }

```

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	$1 \ll 16$	3
998244353	$119 \ll 23$	3
2748779069441	$5 \ll 39$	3
1945555039024054273	$27 \ll 56$	5

Requires: Extended GCD

```

1 template <typename T> struct M {
2     static T MOD; // change to constexpr if already known
3     T v;
4     M(T x = 0) {
5         v = (-MOD <= x && x < MOD) ? x : x % MOD;
6         if (v < 0) v += MOD;
7     }
8     explicit operator T() const { return v; }
9     bool operator==(const M &b) const { return v == b.v; }
10    bool operator!=(const M &b) const { return v != b.v; }
11    M operator-() const { return M(-v); }
12    M operator+(M b) const { return M(v + b.v); }
13    M operator-(M b) const { return M(v - b.v); }
14    M operator*(M b) const { return M((__int128)v * b.v % MOD); }
15    M operator/(M b) const { return *this * (b ^ (MOD - 2)); }
16    // change above implementation to this if MOD is not prime
17    M inv() {
18        auto [p, _, g] = extgcd(v, MOD);
19        return assert(g == 1), p;
20    }
21    friend M operator^(M a, ll b) {
22        M ans(1);
23        for (; b >= 1, a *= a)
24            if (b & 1) ans *= a;
25        return ans;
26    }
27    friend M &operator+=(M &a, M b) { return a = a + b; }
28    friend M &operator-=(M &a, M b) { return a = a - b; }
29    friend M &operator*=(M &a, M b) { return a = a * b; }
30    friend M &operator/=(M &a, M b) { return a = a / b; }
31 };
32 using Mod = M<int>;
33 template <> int Mod::MOD = 1'000'000'007;
34 int &MOD = Mod::MOD;

```

4.1.2. Miller-Rabin

Requires: Mod Struct

```

1 // checks if Mod::MOD is prime
2 bool is_prime() {
3     if (MOD < 2 || MOD % 2 == 0) return MOD == 2;
4     Mod A[] = {2, 7, 61}; // for int values (< 2^31)
5     // ll: 2, 325, 9375, 28178, 450775, 9780504, 1795265022
6     int s = __builtin_ctzll(MOD - 1), i;
7     for (Mod a : A) {
8         Mod x = a ^ (MOD >> s);
9         for (i = 0; i < s && (x + 1).v > 2; i++) x *= x;
10        if (i && x != -1) return 0;
11    }
12    return 1;
13 }

```

4.1.3. Linear Sieve

```

1 constexpr ll MAXN = 1000000;
2 bitset<MAXN> is_prime;
3 vector<ll> primes;
4 ll mpf[MAXN], phi[MAXN], mu[MAXN];
5
6 void sieve() {
7     is_prime.set();
8     is_prime[1] = 0;
9     mu[1] = phi[1] = 1;
10    for (ll i = 2; i < MAXN; i++) {
11        if (is_prime[i]) {
12            mpf[i] = i;
13            primes.push_back(i);
14            phi[i] = i - 1;
15            mu[i] = -1;
16        }
17        for (ll p : primes) {
18            if (p > mpf[i] || i * p >= MAXN) break;
19            is_prime[i * p] = 0;
20            mpf[i * p] = p;
21            mu[i * p] = -mu[i];
22            if (i % p == 0) {
23                phi[i * p] = phi[i] * p, mu[i * p] = 0;
24            }
25            else
26                phi[i * p] = phi[i] * (p - 1);
27        }
28    }
29 }

```

4.1.4. Get Factors

Requires: Linear Sieve

```

1 vector<ll> all_factors(ll n) {
2     vector<ll> fac = {1};
3     while (n > 1) {
4         const ll p = mpf[n];
5         vector<ll> cur = {1};
6         while (n % p == 0) {
7             n /= p;
8             cur.push_back(cur.back() * p);
9         }
10        vector<ll> tmp;
11        for (auto x : fac)
12            for (auto y : cur) tmp.push_back(x * y);
13        tmp.swap(fac);
14    }
15    return fac;
16 }

```

4.1.5. Binary GCD

```

1 // returns the gcd of non-negative a, b
2 ull bin_gcd(ull a, ull b) {
3     if (!a || !b) return a + b;
4     int s = __builtin_ctzll(a | b);
5     a >>= __builtin_ctzll(a);
6     while (b) {
7         if ((b >>= __builtin_ctzll(b)) < a) swap(a, b);
8         b -= a;
9     }
10    return a << s;
11 }

```

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);

```



```

    swap(s -= q * t, t);
    swap(u -= q * v, v);
}
return {s, u, a};
}

```

4.1.7. Chinese Remainder Theorem

Requires: Extended GCD

```

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

```

4.1.8. Baby-Step Giant-Step

Requires: Mod Struct

```

// returns x such that a ^ x = b where x \in [l, r)
ll bsqs(Mod a, Mod b, ll l = 0, ll r = MOD - 1) {
    int m = sqrt(r - l) + 1, i;
    unordered_map<ll, ll> tb;
    Mod d = (a ^ l) / b;
    for (i = 0, d = (a ^ l) / b; i < m; i++, d *= a)
        if (d == 1)
            return l + i;
        else
            tb[(ll)d] = l + i;
    Mod c = Mod(1) / (a ^ m);
    for (i = 0, d = 1; i < m; i++, d *= c)
        if (auto j = tb.find((ll)d); j != tb.end())
            return j->second + i * m;
    return assert(0), -1; // no solution
}

```

4.1.9. Pollard's Rho

```

ll f(ll x, ll mod) { return (x * x + 1) % mod; }
// n should be composite
ll pollard_rho(ll n) {
    if (!(n & 1)) return 2;
    while (1) {
        ll y = 2, x = RNG() % (n - 1) + 1, res = 1;
        for (int sz = 2; res == 1; sz *= 2) {
            for (int i = 0; i < sz && res == 1; i++) {
                x = f(x, n);
                res = __gcd(abs(x - y), n);
            }
            y = x;
        }
        if (res != 0 && res != n) return res;
    }
}

```

4.1.10. Tonelli-Shanks Algorithm

Requires: Mod Struct

```

int legendre(Mod a) {
    if (a == 0) return 0;
    return (a ^ ((MOD - 1) / 2)) == 1 ? 1 : -1;
}
Mod sqrt(Mod a) {
    assert(legendre(a) != -1); // no solution
    ll p = MOD, s = p - 1;
    if (a == 0) return 0;
    if (p == 2) return 1;
    if (p % 4 == 3) return a ^ ((p + 1) / 4);
    int r, m;
    for (r = 0; !(s & 1); r++) s >>= 1;
    Mod n = 2;
    while (legendre(n) != -1) n += 1;
    Mod x = a ^ ((s + 1) / 2), b = a ^ s, g = n ^ s;
    while (b != 1) {
        Mod t = b;
        for (m = 0; t != 1; m++) t *= t;
        Mod gs = g ^ (1LL << (r - m - 1));
        g = gs * gs, x *= gs, b *= g, r = m;
    }
    return x;
}
// to get sqrt(X) modulo p^k, where p is an odd prime:
// c = x^2 (mod p), c = X^2 (mod p^k), q = p^(k-1)
// X = x^q * c^((p^k-2q+1)/2) (mod p^k)

```

4.1.11. Chinese Sieve

```

const ll N = 1000000;
// f, g, h multiplicative, h = f (dirichlet convolution) g
ll pre_g(ll n);
ll pre_h(ll n);
// preprocessed prefix sum of f
ll pre_f[N];
// prefix sum of multiplicative function f
ll solve_f(ll n) {
    static unordered_map<ll, ll> m;
    if (n < N) return pre_f[n];
    if (m.count(n)) return m[n];
    ll ans = pre_h(n);
    for (ll l = 2, r; l <= n; l = r + 1) {
        r = n / (n / l);
        ans -= (pre_g(r) - pre_g(l - 1)) * djs_f(n / l);
    }
    return m[n] = ans;
}

```

4.1.12. Rational Number Binary Search

```

struct QQ {
    ll p, q;
    QQ go(QQ b, ll d) { return {p + b.p * d, q + b.q * d}; }
};
bool pred(QQ);
// returns smallest p/q in [lo, hi] such that
// pred(p/q) is true, and 0 <= p, q <= N
QQ frac_bs(ll N) {
    QQ lo{0, 1}, hi{1, 0};
    if (pred(lo)) return lo;
    assert(pred(hi));
    bool dir = 1, L = 1, H = 1;
    for (; L || H; dir = !dir) {
        ll len = 0, step = 1;
        for (int t = 0; t < 2 && (t ? step /= 2 : step *= 2);)
            if (QQ mid = hi.go(lo, len + step);
                mid.p > N || mid.q > N || dir ^ pred(mid))
                t++;
            else
                len += step;
        swap(lo, hi = hi.go(lo, len));
        (dir ? L : H) = !len;
    }
    return dir ? hi : lo;
}

```

4.1.13. Farey Sequence

```

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

```

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.

```

constexpr int N = 100;
constexpr int INF = 1e9;

struct Matroid {
    Matroid(bitset<N>); // initialize from an independent set
    bool can_add(int); // if adding will break independence
    Matroid remove(int); // removing from the set
};

auto matroid_intersection(int n, const vector<int> &w) {
    bitset<N> S;
    for (int sz = 1; sz <= n; sz++) {
        Matroid M1(S), M2(S);

        vector<vector<pii>> e(n + 2);
        for (int j = 0; j < n; j++)
            if (!S[j]) {
                if (M1.can_add(j)) e[n].emplace_back(j, -w[j]);
                if (M2.can_add(j)) e[j].emplace_back(n + 1, 0);
            }
    }
}

```

```

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
45 if (dis[n + 1].first < INF)
46     for (int x = prev[n + 1]; x != n; x = prev[x])
47         S.flip(x);
48 else
49     break;
50
51 // S is the max-weighted independent set with size sz
52 return S;
53 }

```

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

```

4.3. Algebra

4.3.1. Formal Power Series

```

1 template <typename mint>
2 struct FormalPowerSeries : vector<mint> {
3     using vector<mint>::vector;
4     using FPS = FormalPowerSeries;
5
6     FPS &operator+=(const FPS &r) {
7         if (r.size() > this->size()) this->resize(r.size());
8         for (int i = 0; i < (int)r.size(); i++)
9             (*this)[i] += r[i];
10        return *this;
11    }
12
13    FPS &operator+=(const mint &r) {
14        if (this->empty()) this->resize(1);
15        (*this)[0] += r;
16        return *this;
17    }

```

```

18 FPS &operator-=(const FPS &r) {
19     if (r.size() > this->size()) this->resize(r.size());
20     for (int i = 0; i < (int)r.size(); i++)
21         (*this)[i] -= r[i];
22     return *this;
23 }
24
25 FPS &operator-=(const mint &r) {
26     if (this->empty()) this->resize(1);
27     (*this)[0] -= r;
28     return *this;
29 }
30
31 FPS &operator*=(const mint &v) {
32     for (int k = 0; k < (int)this->size(); k++)
33         (*this)[k] *= v;
34     return *this;
35 }
36
37 FPS &operator/=(const FPS &r) {
38     if (this->size() < r.size()) {
39         this->clear();
40         return *this;
41     }
42     int n = this->size() - r.size() + 1;
43     if ((int)r.size() <= 64) {
44         FPS f(*this), g(r);
45         g.shrink();
46         mint coeff = g.back().inverse();
47         for (auto &x : g) x *= coeff;
48         int deg = (int)f.size() - (int)g.size() + 1;
49         int gs = g.size();
50         FPS quo(deg);
51         for (int i = deg - 1; i >= 0; i--) {
52             quo[i] = f[i + gs - 1];
53             for (int j = 0; j < gs; j++)
54                 f[i + j] -= quo[i] * g[j];
55         }
56         *this = quo * coeff;
57         this->resize(n, mint(0));
58         return *this;
59     }
60     return *this = ((*this).rev().pre(n) * r.rev().inv(n))
61                     .pre(n)
62                     .rev();
63 }
64
65 FPS &operator%=(const FPS &r) {
66     *this -= *this / r * r;
67     shrink();
68     return *this;
69 }
70
71 FPS operator+(const FPS &r) const {
72     return FPS(*this) += r;
73 }
74
75 FPS operator+(const mint &v) const {
76     return FPS(*this) += v;
77 }
78
79 FPS operator-(const FPS &r) const {
80     return FPS(*this) -= r;
81 }
82
83 FPS operator-(const mint &v) const {
84     return FPS(*this) -= v;
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 }
114
115 FPS rev() const {

```

```

109     FPS ret(*this);
110     reverse(begin(ret), end(ret));
111     return ret;
112 }
113
114 FPS dot(FPS r) const {
115     FPS ret(min(this->size(), r.size()));
116     for (int i = 0; i < (int)ret.size(); i++)
117         ret[i] = (*this)[i] * r[i];
118     return ret;
119 }
120
121 FPS pre(int sz) const {
122     return FPS(begin(*this),
123               begin(*this) + min((int)this->size(), sz));
124 }
125
126 FPS operator>>(int sz) const {
127     if ((int)this->size() <= sz) return {};
128     FPS ret(*this);
129     ret.erase(ret.begin(), ret.begin() + sz);
130     return ret;
131 }
132
133 FPS operator<<(int sz) const {
134     FPS ret(*this);
135     ret.insert(ret.begin(), sz, mint(0));
136     return ret;
137 }
138
139 FPS diff() const {
140     const int n = (int)this->size();
141     FPS ret(max(0, n - 1));
142     mint one(1), coeff(1);
143     for (int i = 1; i < n; i++) {
144         ret[i - 1] = (*this)[i] * coeff;
145         coeff += one;
146     }
147     return ret;
148 }
149
150 FPS integral() const {
151     const int n = (int)this->size();
152     FPS ret(n + 1);
153     ret[0] = mint(0);
154     if (n > 0) ret[1] = mint(1);
155     auto mod = mint::get_mod();
156     for (int i = 2; i <= n; i++)
157         ret[i] = (-ret[mod % i]) * (mod / i);
158     for (int i = 0; i < n; i++) ret[i + 1] *= (*this)[i];
159     return ret;
160 }
161
162 mint eval(mint x) const {
163     mint r = 0, w = 1;
164     for (auto &v : *this) r += w * v, w *= x;
165     return r;
166 }
167
168 FPS log(int deg = -1) const {
169     assert((*this)[0] == mint(1));
170     if (deg == -1) deg = (int)this->size();
171     return (this->diff() * this->inv(deg))
172         .pre(deg - 1)
173         .integral();
174 }
175
176 FPS pow(int64_t k, int deg = -1) const {
177     const int n = (int)this->size();
178     if (deg == -1) deg = n;
179     for (int i = 0; i < n; i++) {
180         if ((*this)[i] != mint(0)) {
181             if (i * k > deg) return FPS(deg, mint(0));
182             mint rev = mint(1) / (*this)[i];
183             FPS ret =
184                 (((*this * rev) >> i).log(deg) * k).exp(deg) *
185                 ((*this)[i].pow(k));
186             ret = (ret << (i * k)).pre(deg);
187             if ((int)ret.size() < deg) ret.resize(deg, mint(0));
188             return ret;
189         }
190     }
191     return FPS(deg, mint(0));
192 }
193
194 static void *ntt_ptr;
195 static void set_fft();
196 FPS &operator*=(const FPS &r);
197 void ntt();
198 void intt();

```

```

199 void ntt_doubling();
200 static int ntt_ptr();
201 FPS inv(int deg = -1) const;
202 FPS exp(int deg = -1) const;
203 };
204 template <typename mint>
205 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 :

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

Then

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

5. Randomisation

5.1. Simulated Annealing

```

1 double get_rand_double() {
2     return double(
3         uniform_int_distribution<int>(0, INF)(rng)) /
4         INF;
5 }
6
7 int main() {
8     const int iterations = 10000000;
9     double mpl = pow((1e-5) / temp, (double)1 / iterations);
10    for (int i = 0; i < iterations; ++i) {
11        t *= mpl;
12        if (new_score >= score ||
13            get_rand_double() <
14            exp((new_score - score) / temp)) {
15            score = new_score;
16            return;
17        }
18    }
19 }

```

6. Numeric

6.1. 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 };
```

6.2. 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 }
```

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

6.4. Fast Walsh-Hadamard Transform

Requires: Mod Struct

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

6.5. Subset Convolution

Requires: Mod Struct

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

6.6. Linear Recurrences

6.6.1. 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 }
```

6.6.2. 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 >= 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 };
```

6.7. Matrices

6.7.1. Determinant

Requires: Mod Struct

```

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

```

```

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

```

6.7.2. Inverse

```

1 // Returns rank.
2 // Result is stored in A unless singular (rank < n).
3 // For prime powers, repeatedly set
4 // A^{-1} = A^{-1} (2I - A*A^{-1}) (mod p^k)
5 // where A^{-1} starts as the inverse of A mod p,
6 // and k is doubled in each step.
7
8 int matInv(vector<vector<double>> &A) {
9     int n = sz(A);
10    vi col(n);
11    vector<vector<double>> tmp(n, vector<double>(n));
12    rep(i, 0, n) tmp[i][i] = 1, col[i] = i;
13
14    rep(i, 0, n) {
15        int r = i, c = i;
16        rep(j, i, n)
17            rep(k, i, n) if (fabs(A[j][k]) > fabs(A[r][c])) r = j, c = k;
18
19        if (fabs(A[r][c]) < 1e-12) return i;
20        A[i].swap(A[r]);
21        tmp[i].swap(tmp[r]);
22        rep(j, 0, n) swap(A[j][i], A[j][c]),
23            swap(tmp[j][i], tmp[j][c]);
24        swap(col[i], col[c]);
25        double v = A[i][i];
26        rep(j, i + 1, n) {
27            double f = A[j][i] / v;
28            A[j][i] = 0;
29            rep(k, i + 1, n) A[j][k] -= f * A[i][k];
30            rep(k, 0, n) tmp[j][k] -= f * tmp[i][k];
31        }
32        rep(j, i + 1, n) A[i][j] /= v;
33        rep(j, 0, n) tmp[i][j] /= v;
34        A[i][i] = 1;
35    }
36
37    for (int i = n - 1; i > 0; --i) rep(j, 0, i) {
38        double v = A[j][i];

```

```

        rep(k, 0, n) tmp[j][k] -= v * tmp[i][k];
    }
}

rep(i, 0, n) rep(j, 0, n) A[col[i]][col[j]] = tmp[i][j];
return n;
}

int matInv_mod(vector<vector<ll>> &A) {
    int n = sz(A);
    vi col(n);
    vector<vector<ll>> tmp(n, vector<ll>(n));
    rep(i, 0, n) tmp[i][i] = 1, col[i] = i;

    rep(i, 0, n) {
        int r = i, c = i;
        rep(j, i, n) rep(k, i, n) if (A[j][k]) {
            r = j;
            c = k;
            goto found;
        }
        return i;
    found:
        A[i].swap(A[r]);
        tmp[i].swap(tmp[r]);
        rep(j, 0, n) swap(A[j][i], A[j][c]),
            swap(tmp[j][i], tmp[j][c]);
        swap(col[i], col[c]);
        ll v = modpow(A[i][i], mod - 2);
        rep(j, i + 1, n) {
            ll f = A[j][i] * v % mod;
            A[j][i] = 0;
            rep(k, i + 1, n) A[j][k] =
                (A[j][k] - f * A[i][k]) % mod;
            rep(k, 0, n) tmp[j][k] =
                (tmp[j][k] - f * tmp[i][k]) % mod;
        }
        rep(j, i + 1, n) A[i][j] = A[i][j] * v % mod;
        rep(j, 0, n) tmp[i][j] = tmp[i][j] * v % mod;
        A[i][i] = 1;
    }

    for (int i = n - 1; i > 0; --i) rep(j, 0, i) {
        ll v = A[j][i];
        rep(k, 0, n) tmp[j][k] =
            (tmp[j][k] - v * tmp[i][k]) % mod;
    }

    rep(i, 0, n) rep(j, 0, n) A[col[i]][col[j]] =
        tmp[i][j] % mod + (tmp[i][j] < 0 ? mod : 0);
    return n;
}

```

6.7.3. Characteristic Polynomial

```

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

```



```

37     for (int m = 1; m < i; m++) {
38         x += -a[i - m][i - m - 1];
39         T coe = x * a[i - m - 1][i - 1];
40         for (int j = 0; j < i - m; j++)
41             p[i][j] += coe * p[i - m - 1][j];
42     }
43 }
44 return p[N];
45 }

```

6.7.4. Solve Linear Equation

```

1  typedef vector<double> vd;
2  const double eps = 1e-12;
3
4  // solves for x: A * x = b
5  int solveLinear(vector<vd> &A, vd &b, vd &x) {
6      int n = sz(A), m = sz(x), rank = 0, br, bc;
7      if (n) assert(sz(A[0]) == m);
8      vi col(m);
9      iota(all(col), 0);
10
11     rep(i, 0, n) {
12         double v, bv = 0;
13         rep(r, i, n) rep(c, i, m) if ((v = fabs(A[r][c])) > bv)
14             br = r, bc = c, bv = v;
15         if (bv <= eps) {
16             rep(j, i, n) if (fabs(b[j]) > eps) return -1;
17             break;
18         }
19         swap(A[i], A[br]);
20         swap(b[i], b[br]);
21         swap(col[i], col[bc]);
22         rep(j, 0, n) swap(A[j][i], A[j][bc]);
23         bv = 1 / A[i][i];
24         rep(j, i + 1, n) {
25             double fac = A[j][i] * bv;
26             b[j] -= fac * b[i];
27             rep(k, i + 1, m) A[j][k] -= fac * A[i][k];
28         }
29         rank++;
30     }
31
32     x.assign(m, 0);
33     for (int i = rank; i--;) {
34         b[i] /= A[i][i];
35         x[col[i]] = b[i];
36         rep(j, 0, i) b[j] -= A[j][i] * b[i];
37     }
38     return rank; // (multiple solutions if rank < m)
39 }

```

6.8. Polynomial Interpolation

```

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

```

6.9. Simplex Algorithm

```

1  // Two-phase simplex algorithm for solving linear programs
2  // of the form
3  //
4  //      maximize      c^T x
5  //      subject to    Ax <= b
6  //                   x >= 0
7  //
8  // INPUT: A -- an m x n matrix
9  //         b -- an m-dimensional vector
10 //         c -- an n-dimensional vector
11 //         x -- a vector where the optimal solution will be
12 //             stored
13 //
14 // OUTPUT: value of the optimal solution (infinity if
15 // unbounded
16 //         above, nan if infeasible)

```

```

17 //
18 // To use this code, create an LPSolver object with A, b,
19 // and c as arguments. Then, call Solve(x).
20
21 typedef long double ld;
22 typedef vector<ld> vd;
23 typedef vector<vd> vvd;
24 typedef vector<int> vi;
25
26 const ld EPS = 1e-9;
27
28 struct LPSolver {
29     int m, n;
30     vi B, N;
31     vvd D;
32
33     LPSolver(const vvd &A, const vd &b, const vd &c)
34         : m(b.size()), n(c.size()), N(n + 1), B(m),
35           D(m + 2, vd(n + 2)) {
36         for (int i = 0; i < m; i++)
37             for (int j = 0; j < n; j++) D[i][j] = A[i][j];
38         for (int i = 0; i < m; i++) {
39             B[i] = n + i;
40             D[i][n] = -1;
41             D[i][n + 1] = b[i];
42         }
43         for (int j = 0; j < n; j++) {
44             N[j] = j;
45             D[m][j] = -c[j];
46         }
47         N[n] = -1;
48         D[m + 1][n] = 1;
49     }
50
51     void Pivot(int r, int s) {
52         double inv = 1.0 / D[r][s];
53         for (int i = 0; i < m + 2; i++)
54             if (i != r)
55                 for (int j = 0; j < n + 2; j++)
56                     if (j != s) D[i][j] -= D[r][j] * D[i][s] * inv;
57         for (int j = 0; j < n + 2; j++)
58             if (j != s) D[r][j] *= inv;
59         for (int i = 0; i < m + 2; i++)
60             if (i != r) D[i][s] *= -inv;
61         D[r][s] = inv;
62         swap(B[r], N[s]);
63     }
64
65     bool Simplex(int phase) {
66         int x = phase == 1 ? m + 1 : m;
67         while (true) {
68             int s = -1;
69             for (int j = 0; j <= n; j++) {
70                 if (phase == 2 && N[j] == -1) continue;
71                 if (s == -1 || D[x][j] < D[x][s] ||
72                     D[x][j] == D[x][s] && N[j] < N[s])
73                     s = j;
74             }
75             if (D[x][s] > -EPS) return true;
76             int r = -1;
77             for (int i = 0; i < m; i++) {
78                 if (D[i][s] < EPS) continue;
79                 if (r == -1 ||
80                     D[i][n + 1] / D[i][s] < D[r][n + 1] / D[r][s] ||
81                     (D[i][n + 1] / D[i][s] ==
82                      D[r][n + 1] / D[r][s]) &&
83                     B[i] < B[r])
84                     r = i;
85             }
86             if (r == -1) return false;
87             Pivot(r, s);
88         }
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         }
109     }
110 }

```

```

107     }
109     if (!Simplex(2)) return numeric_limits<ld>::infinity();
110     x = vd(n);
111     for (int i = 0; i < m; i++)
112         if (B[i] < n) x[B[i]] = D[i][n + 1];
113     return D[m][n + 1];
114 }
115 };
117 int main() {
118
119     const int m = 4;
120     const int n = 3;
121     ld _A[m][n] = {
122         {6, -1, 0}, {-1, -5, 0}, {1, 5, 1}, {-1, -5, -1}};
123     ld _b[m] = {10, -4, 5, -5};
124     ld _c[n] = {1, -1, 0};
125
126     vvd A(m);
127     vd b(_b, _b + m);
128     vd c(_c, _c + n);
129     for (int i = 0; i < m; i++) A[i] = vd(_A[i], _A[i] + n);
130
131     LPSolver solver(A, b, c);
132     vd x;
133     ld value = solver.Solve(x);
134
135     cerr << "VALUE: " << value << endl; // VALUE: 1.29032
136     cerr << "SOLUTION:"; // SOLUTION: 1.74194 0.451613 1
137     for (size_t i = 0; i < x.size(); i++) cerr << " " << x[i];
138     cerr << endl;
139     return 0;
140 }

```

7. Geometry

7.1. Point

```

1 template <typename T> struct P {
2     T x, y;
3     P(T x = 0, T y = 0) : x(x), y(y) {}
4     bool operator<(const P &p) const {
5         return tie(x, y) < tie(p.x, p.y);
6     }
7     bool operator==(const P &p) const {
8         return tie(x, y) == tie(p.x, p.y);
9     }
10    P operator-() const { return {-x, -y}; }
11    P operator+(P p) const { return {x + p.x, y + p.y}; }
12    P operator-(P p) const { return {x - p.x, y - p.y}; }
13    P operator*(T d) const { return {x * d, y * d}; }
14    P operator/(T d) const { return {x / d, y / d}; }
15    T dist2() const { return x * x + y * y; }
16    double len() const { return sqrt(dist2()); }
17    P unit() const { return *this / len(); }
18    friend T dot(P a, P b) { return a.x * b.x + a.y * b.y; }
19    friend T cross(P a, P b) { return a.x * b.y - a.y * b.x; }
20    friend T cross(P a, P b, P o) {
21        return cross(a - o, b - o);
22    }
23 };
24 using pt = P<ll>;

```

7.1.1. 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    }
60 };

```

7.1.2. Spherical Coordinates

```

1 struct car_p {
2     double x, y, z;
3 };
4 struct sph_p {
5     double r, theta, phi;
6 };
7
8 sph_p conv(car_p p) {
9     double r = sqrt(p.x * p.x + p.y * p.y + p.z * p.z);
10    double theta = asin(p.y / r);
11    double phi = atan2(p.y, p.x);
12    return {r, theta, phi};
13 }
14 car_p conv(sph_p p) {
15     double x = p.r * cos(p.theta) * sin(p.phi);
16     double y = p.r * cos(p.theta) * cos(p.phi);
17     double z = p.r * sin(p.theta);
18     return {x, y, z};
19 }

```

7.2. 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 }

```

7.3. 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) {

```

```

11     while (t > s + 1 && cross(i, h[t - 1], h[t - 2]) >= 0)
12         t--;
13     h[t++] = i;
14     return h.resize(t), h;
15 }

```

7.3.1. 3D Hull

```

1 typedef Point3D<double> P3;
2
3 struct PR {
4     void ins(int x) { (a == -1 ? a : b) = x; }
5     void rem(int x) { (a == x ? a : b) = -1; }
6     int cnt() { return (a != -1) + (b != -1); }
7     int a, b;
8 };
9
10 struct F {
11     P3 q;
12     int a, b, c;
13 };
14
15 vector<F> hull3d(const vector<P3> &A) {
16     assert(sz(A) >= 4);
17     vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1, -1}));
18     #define E(x, y) E[f.x][f.y]
19     vector<F> FS;
20     auto mf = [&](int i, int j, int k, int l) {
21         P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
22         if (q.dot(A[l]) > q.dot(A[i])) q = q * -1;
23         F f{q, i, j, k};
24         E(a, b).ins(k);
25         E(a, c).ins(j);
26         E(b, c).ins(i);
27         FS.push_back(f);
28     };
29     rep(i, 0, 4) rep(j, i + 1, 4) rep(k, j + 1, 4)
30     mf(i, j, k, 6 - i - j - k);
31
32     rep(i, 4, sz(A)) {
33         rep(j, 0, sz(FS)) {
34             F f = FS[j];
35             if (f.q.dot(A[i]) > f.q.dot(A[f.a])) {
36                 E(a, b).rem(f.c);
37                 E(a, c).rem(f.b);
38                 E(b, c).rem(f.a);
39                 swap(FS[j--], FS.back());
40                 FS.pop_back();
41             }
42         }
43         int nw = sz(FS);
44         rep(j, 0, nw) {
45             F f = FS[j];
46             #define C(a, b, c)
47             if (E(a, b).cnt() != 2) mf(f.a, f.b, i, f.c);
48             C(a, b, c);
49             C(a, c, b);
50             C(b, c, a);
51         }
52     }
53     for (F &it : FS)
54         if ((A[it.b] - A[it.a])
55             .cross(A[it.c] - A[it.a])
56             .dot(it.q) <= 0)
57             swap(it.c, it.b);
58     return FS;
59 }

```

7.4. 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 }

```

7.5. 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     };
9 }

```

```

10 };
11 rotate(c.begin(), min_element(ALL(c), rcmp), c.end());
12 c.push_back(c[0]);
13 vector<pt> ret;
14 for (int i = 1; i < c.size(); i++)
15     ret.push_back(c[i] - c[i - 1]);
16 return ret;
17 };
18 auto dp = diff(p), dq = diff(q);
19 pt cur = p[0] + q[0];
20 vector<pt> d(dp.size() + dq.size(), ret = {cur};
21 // include angle_cmp from angular-sort.cpp
22 merge(ALL(dp), ALL(dq), d.begin(), angle_cmp);
23 // optional: make ret strictly convex (UB if degenerate)
24 int now = 0;
25 for (int i = 1; i < d.size(); i++) {
26     if (cross(d[i], d[now]) == 0)
27         d[now] = d[now] + d[i];
28     else
29         d[++now] = d[i];
30 }
31 d.resize(now + 1);
32 // end optional part
33 for (pt v : d) ret.push_back(cur = cur + v);
34 return ret.pop_back(), ret;
35 }

```

7.6. 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 }

```

7.6.1. 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     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)
12            l = m;
13        else if (a < 0)
14            r = m;
15        else
16            return dot(c[0] - p, c[m] - p) <= 0;
17    }
18    if (l == r)
19        return dot(c[0] - p, c[l] - p) <= 0;
20    else
21        return cross(c[l], c[r], p) >= 0;
22 }
23
24 // with preprocessing version
25 vector<pt> vecs;
26 pt center;
27 // p must be a strict convex hull, counterclockwise
28 // BEWARE OF OVERFLOWS!!
29 void preprocess(vector<pt> &p) {
30     for (auto &v : p) v = v * 3;
31     center = p[0] + p[1] + p[2];
32     center.x /= 3, center.y /= 3;
33     for (auto &v : p) v = v - center;
34     vecs = (angular_sort(p), p);
35 }
36
37 bool intersect_strict(pt a, pt b, pt c, pt d) {
38     if (cross(b, c, a) * cross(b, d, a) > 0) return false;
39     if (cross(d, a, c) * cross(d, b, c) >= 0) return false;
40     return true;
41 }
42 // if point is inside or on border
43 bool query(pt p) {
44     p = p * 3 - center;
45     auto pr = upper_bound(ALL(vecs), p, angle_cmp);
46 }

```

```

45 if (pr == vecs.end()) pr = vecs.begin();
46 auto pl = (pr == vecs.begin()) ? vecs.back() : *(pr - 1);
47 return !intersect_strict({0, 0}, p, pl, *pr);
48 }

```

7.6.2. Offline Multiple Points Version

Requires: GNU PBDS, Point

```

1 using Double = __float128;
2 using Point = pt<Double, Double>;
3
4 int n, m;
5 vector<Point> poly;
6 vector<Point> query;
7 vector<int> ans;
8
9 struct Segment {
10     Point a, b;
11     int id;
12 };
13 vector<Segment> segs;
14
15 Double Xnow;
16 inline Double get_y(const Segment &u, Double xnow = Xnow) {
17     const Point &a = u.a;
18     const Point &b = u.b;
19     return (a.y * (b.x - xnow) + b.y * (xnow - a.x)) /
20         (b.x - a.x);
21 }
22 bool operator<(Segment u, Segment v) {
23     Double yu = get_y(u);
24     Double yv = get_y(v);
25     if (yu != yv) return yu < yv;
26     return u.id < v.id;
27 }
28 ordered_map<Segment> st;
29
30 struct Event {
31     int type; // +1 insert seg, -1 remove seg, 0 query
32     Double x, y;
33     int id;
34 };
35 bool operator<(Event a, Event b) {
36     if (a.x != b.x) return a.x < b.x;
37     if (a.type != b.type) return a.type < b.type;
38     return a.y < b.y;
39 }
40 vector<Event> events;
41
42 void solve() {
43     set<Double> xs;
44     set<Point> ps;
45     for (int i = 0; i < n; i++) {
46         xs.insert(poly[i].x);
47         ps.insert(poly[i]);
48     }
49     for (int i = 0; i < n; i++) {
50         Segment s{poly[i], poly[(i + 1) % n], i};
51         if (s.a.x > s.b.x ||
52             (s.a.x == s.b.x && s.a.y > s.b.y)) {
53             swap(s.a, s.b);
54         }
55         segs.push_back(s);
56     }
57     if (s.a.x != s.b.x) {
58         events.push_back({+1, s.a.x + 0.2, s.a.y, i});
59         events.push_back({-1, s.b.x - 0.2, s.b.y, i});
60     }
61 }
62 for (int i = 0; i < m; i++) {
63     events.push_back({0, query[i].x, query[i].y, i});
64 }
65 sort(events.begin(), events.end());
66 int cnt = 0;
67 for (Event e : events) {
68     int i = e.id;
69     Xnow = e.x;
70     if (e.type == 0) {
71         Double x = e.x;
72         Double y = e.y;
73         Segment tmp = {{x - 1, y}, {x + 1, y}, -1};
74         auto it = st.lower_bound(tmp);
75
76         if (ps.count(query[i]) > 0) {
77             ans[i] = 0;
78         } else if (xs.count(x) > 0) {
79             ans[i] = -2;
80         } else if (it != st.end() &&
81             get_y(*it) == get_y(tmp)) {
82             ans[i] = 0;
83         }
84     }
85 }

```

```

83     } else if (it != st.begin() &&
84         get_y(*prev(it)) == get_y(tmp)) {
85         ans[i] = 0;
86     } else {
87         int rk = st.order_of_key(tmp);
88         if (rk % 2 == 1) {
89             ans[i] = 1;
90         } else {
91             ans[i] = -1;
92         }
93     }
94 } else if (e.type == 1) {
95     st.insert(segs[i]);
96     assert((int)st.size() == ++cnt);
97 } else if (e.type == -1) {
98     st.erase(segs[i]);
99     assert((int)st.size() == --cnt);
100 }
101 }

```

7.7. 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 }

```

7.8. Minimum Enclosing Circle

```

1 typedef Point<double> P;
2 double ccRadius(const P &A, const P &B, const P &C) {
3     return (B - A).dist() * (C - B).dist() * (A - C).dist() /
4         abs((B - A).cross(C - A)) / 2;
5 }
6 P ccCenter(const P &A, const P &B, const P &C) {
7     P b = C - A, c = B - A;
8     return A + (b * c.dist2() - c * b.dist2()).perp() /
9         b.cross(c) / 2;
10 }
11 pair<P, double> mec(vector<P> ps) {
12     shuffle(all(ps), mt19937(time(0)));
13     P o = ps[0];
14     double r = 0, EPS = 1 + 1e-8;
15     rep(i, 0, sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
16         o = ps[i], r = 0;
17     }
18     rep(j, 0, i) if ((o - ps[j]).dist() > r * EPS) {
19         o = (ps[i] + ps[j]) / 2;
20         r = (o - ps[i]).dist();
21     }
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 return {o, r};

```

7.9. Delaunay Triangulation

```

1 typedef Point<ll> P;
2 typedef struct Quad *Q;
3 typedef __int128_t lll; // (can be ll if coords are < 2e4)
4 P arb(LLONG_MAX, LLONG_MAX); // not equal to any other point
5
6 struct Quad {
7     bool mark;
8     Q o, rot;
9     P p;
10     P F() { return r()->p; }
11     Q r() { return rot->rot; }
12     Q prev() { return rot->o->rot; }
13     Q next() { return r()->prev(); }

```

```

};
15 bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
17     lll p2 = p.dist2(), A = a.dist2() - p2,
        B = b.dist2() - p2, C = c.dist2() - p2;
19     return p.cross(a, b) * C + p.cross(b, c) * A +
        p.cross(c, a) * B >
21         0;
23 }
24 Q makeEdge(P orig, P dest) {
    Q q[] = {new Quad{0, 0, 0, orig}, new Quad{0, 0, 0, arb},
25             new Quad{0, 0, 0, dest}, new Quad{0, 0, 0, arb}};
    rep(i, 0, 4) q[i]->o = q[-i & 3],
27             q[i]->rot = q[(i + 1) & 3];
    return *q;
29 }
30 void splice(Q a, Q b) {
    swap(a->o->rot->o, b->o->rot->o);
    swap(a->o, b->o);
33 }
34 Q connect(Q a, Q b) {
    Q q = makeEdge(a->F(), b->p);
    splice(q, a->next());
    splice(q->r(), b);
    return q;
39 }

41 pair<Q, Q> rec(const vector<P> &s) {
    if (sz(s) <= 3) {
43         Q a = makeEdge(s[0], s[1]),
            b = makeEdge(s[1], s.back());
45         if (sz(s) == 2) return {a, a->r()};
        splice(a->r(), b);
47         auto side = s[0].cross(s[1], s[2]);
        Q c = side ? connect(b, a) : 0;
49         return {side < 0 ? c->r() : a, side < 0 ? c : b->r()};
    }

51 #define H(e) e->F(), e->p
52 #define valid(e) (e->F().cross(H(base)) > 0)
    Q A, B, ra, rb;
55     int half = sz(s) / 2;
    tie(ra, A) = rec({all(s) - half});
    tie(B, rb) = rec({sz(s) - half + all(s)});
57     while ((B->p.cross(H(A)) < 0 && (A = A->next())) ||
        (A->p.cross(H(B)) > 0 && (B = B->r()->o)))
59         ;
    Q base = connect(B->r(), A);
    if (A->p == ra->p) ra = base->r();
    if (B->p == rb->p) rb = base;
63

65 #define DEL(e, init, dir)
    Q e = init->dir;
67     if (valid(e))
        while (circ(e->dir->F(), H(base), e->F())) {
69         Q t = e->dir;
            splice(e, e->prev());
71         splice(e->r(), e->r()->prev());
            e = t;
73     }
    for (;;) {
75         DEL(LC, base->r(), o);
        DEL(RC, base, prev());
77         if (!valid(LC) && !valid(RC)) break;
        if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
79             base = connect(RC, base->r());
        else
81             base = connect(base->r(), LC->r());
    }
83     return {ra, rb};
85 }

87 // returns [A_0, B_0, C_0, A_1, B_1, ...]
88 // where A_i, B_i, C_i are counter-clockwise triangles
vector<P> triangulate(vector<P> pts) {
89     sort(all(pts));
    assert(unique(all(pts)) == pts.end());
91     if (sz(pts) < 2) return {};
    Q e = rec(pts).first;
    vector<Q> q = {e};
    int qi = 0;
95     while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
    #define ADD
97     {
        Q c = e;
99         do {
            c->mark = 1;
101             pts.push_back(c->p);
            q.push_back(c->r());
            c = c->next();
103

```

```

        } while (c != e);
105     }
    ADD;
107     pts.clear();
    while (qi < sz(q))
109         if (!(e = q[qi++])->mark) ADD;
    return pts;
111 }

```

7.9.1. Slower Version

```

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

```

7.10. Half Plane Intersection

```

1 struct Line {
    Point P;
    Vector v;
3     bool operator<(const Line &b) const {
        return atan2(v.y, v.x) < atan2(b.v.y, b.v.x);
    }
};
5 bool OnLeft(const Line &L, const Point &p) {
    return Cross(L.v, p - L.P) > 0;
7 }
8 Point GetIntersection(Line a, Line b) {
    Vector u = a.P - b.P;
13     Double t = Cross(b.v, u) / Cross(a.v, b.v);
    return a.P + a.v * t;
15 }
16 int HalfplaneIntersection(Line *L, int n, Point *poly) {
    sort(L, L + n);
17
19     int first, last;
    Point *p = new Point[n];
    Line *q = new Line[n];
    q[first = last = 0] = L[0];
23     for (int i = 1; i < n; i++) {
        while (first < last && !OnLeft(L[i], p[last - 1]))
25         last--;
        while (first < last && !OnLeft(L[i], p[first])) first++;
27         q[++last] = L[i];
        if (fabs(Cross(q[last].v, q[last - 1].v)) < EPS) {
29             last--;
            if (OnLeft(q[last], L[i].P)) q[last] = L[i];
31         }
        if (first < last)
33             p[last - 1] = GetIntersection(q[last - 1], q[last]);
    }
35     while (first < last && !OnLeft(q[first], p[last - 1]))
        last--;
37     if (last - first <= 1) return 0;
    p[last] = GetIntersection(q[last], q[first]);
39
41     int m = 0;
    for (int i = first; i <= last; i++) poly[m++] = p[i];
    return m;
43 }

```

8. Strings

8.1. Knuth-Morris-Pratt Algorithm

```

1 vector<int> pi(const string &s) {
    vector<int> p(s.size());
3     for (int i = 1; i < s.size(); i++) {
        int g = p[i - 1];
5         while (g && s[i] != s[g]) g = p[g - 1];
        p[i] = g + (s[i] == s[g]);
7     }
    return p;
9 }
10 vector<int> match(const string &s, const string &pat) {
    vector<int> p = pi(pat + '\0' + s), res;
11     for (int i = p.size() - s.size(); i < p.size(); i++)

```



```

13     if (p[i] == pat.size())
14         res.push_back(i - 2 * pat.size());
15     return res;
16 }

```

8.2. 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;
56        get_node(1);
57    }
58 } AC;
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     }
75 }

```

8.3. Suffix Array

```

1 // sa[i]: starting index of suffix at rank i
2 // 0-indexed, sa[0] = n (empty string)
3 // lcp[i]: lcp of sa[i] and sa[i - 1], lcp[0] = 0
4 struct SuffixArray {
5     vector<int> sa, lcp;

```

```

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

```

8.4. 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)
22            last->green = root;
23        else {
24            Node *pot_green = p->edge[c];
25            if ((pot_green->max_len) == (p->max_len + 1))
26                last->green = pot_green;
27            else {
28                Node *wish = get_node(p->max_len + 1);
29                wish->times = 0;
30                while (p && p->edge[c] == pot_green)
31                    p->edge[c] = wish, p = p->green;
32                for (int i = 0; i < maxc; i++)
33                    wish->edge[i] = pot_green->edge[i];
34                wish->green = pot_green->green;
35                pot_green->green = wish;
36                last->green = wish;
37            }
38        }
39    }
40    Node *q[maxn * 2];
41    int ql, qr;
42    void get_times(Node *p) {
43        ql = 0, qr = -1, reg[0].in = 1;
44        for (int i = 1; i < top; i++) reg[i].green->in++;
45        for (int i = 0; i < top; i++)
46            if (!reg[i].in) q[++qr] = &reg[i];
47        while (ql <= qr) {
48            q[ql]->green->times += q[ql]->times;
49            if (!(--q[ql]->green->in)) q[++qr] = q[ql]->green;
50            ql++;
51        }
52    }
53    void build(const string &s) {
54        top = 0;
55        root = last = get_node(0);
56        for (char c : s) insert(c - 'a'); // change char id
57        get_times(root);

```

```

}
// call build before solve
int solve(const string &s) {
    Node *p = root;
    for (char c : s)
        if (!(p = p->edge[c - 'a'])) // change char id
            return 0;
    return p->times;
}
};

```

8.5. Z Value

```

1 int z[n];
void zval(string s) {
3 // z[i] => longest common prefix of s and s[i:], i > 0
    int n = s.size();
    z[0] = 0;
    for (int b = 0, i = 1; i < n; i++) {
6         if (z[b] + b <= i)
            z[i] = 0;
7         else
            z[i] = min(z[i - b], z[b] + b - i);
11        while (s[i + z[i]] == s[z[i]]) z[i]++;
            if (i + z[i] > b + z[b]) b = i;
13    }
}

```

8.6. Manacher's Algorithm

```

1 int z[n];
void manacher(string s) {
3 // z[i] => longest odd palindrome centered at i is
// s[i - z[i] ... i + z[i]]
5 // to get all palindromes (including even length),
// insert a '#' between each s[i] and s[i + 1]
    int n = s.size();
    z[0] = 0;
    for (int b = 0, i = 1; i < n; i++) {
6         if (z[b] + b >= i)
            z[i] = min(z[2 * b - i], b + z[b] - i);
7         else
            z[i] = 0;
11        while (i + z[i] + 1 < n && i - z[i] - 1 >= 0 &&
            s[i + z[i] + 1] == s[i - z[i] - 1])
            z[i]++;
17        if (z[i] + i > z[b] + b) b = i;
    }
19 }

```

8.7. Minimum Rotation

```

1 int min_rotation(string s) {
    int a = 0, n = s.size();
    s += s;
3    for (int b = 0; b < n; b++) {
        for (int k = 0; k < n; k++) {
5            if (a + k == b || s[a + k] < s[b + k]) {
                b += max(0, k - 1);
                break;
            }
11           if (s[a + k] > s[b + k]) {
                a = b;
                break;
            }
13        }
    }
15    return a;
17 }

```

8.8. Palindromic Tree

```

1 struct palindromic_tree {
    struct node {
3        int next[26], fail, len;
        int cnt,
5        num; // cnt: appear times, num: number of pal. suf.
        node(int l = 0) : fail(0), len(l), cnt(0), num(0) {
7            for (int i = 0; i < 26; ++i) next[i] = 0;
        }
    };
    vector<node> St;
11    vector<char> s;
    int last, n;
    palindromic_tree() : St(2), last(1), n(0) {
13        St[0].fail = 1, St[1].len = -1, s.pb(-1);
    }
    inline void clear() {
15        St.clear(), s.clear(), last = 1, n = 0;
        St.pb(0), St.pb(-1);
17    }
}

```

```

19    St[0].fail = 1, s.pb(-1);
}
21 inline int get_fail(int x) {
    while (s[n - St[x].len - 1] != s[n]) x = St[x].fail;
23    return x;
}
25 inline void add(int c) {
    s.push_back(c - 'a'), ++n;
27    int cur = get_fail(last);
    if (!St[cur].next[c]) {
29        int now = SZ(St);
        St.pb(St[cur].len + 2);
        St[now].fail = St[get_fail(St[cur].fail)].next[c];
        St[cur].next[c] = now;
        St[now].num = St[St[now].fail].num + 1;
31    }
    last = St[cur].next[c], ++St[last].cnt;
33 }
35 inline void count() { // counting cnt
    auto i = St.rbegin();
37    for (; i != St.rend(); ++i) {
        St[i->fail].cnt += i->cnt;
41    }
}
43 inline int size() { // The number of diff. pal.
    return SZ(St) - 2;
45 }
};

```

9. 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.
 - 5 - Submit the correct file.
- 7 - General Debugging:
 - Read the whole problem again.
 - Have a teammate read the problem.
 - Have a teammate read your code.
 - 11 - Explain your solution to them (or a rubber duck).
 - Print the code and its output / debug output.
 - 13 - Go to the toilet.
- 15 - Wrong Answer:
 - Any possible overflows?
 - > `__int128`?
 - Try `-ftrapv` or `#pragma GCC optimize("trapv")`
 - 17 - Floating point errors?
 - > `long double`?
 - turn off math optimizations
 - check for `'=='`, `'>='`, `'acos(1.000000001)'`, etc.
 - 21 - Did you forget to sort or unique?
 - Generate large and worst "corner" cases.
 - 23 - Check your `'m' / 'n'`, `'i' / 'j'` and `'x' / 'y'`.
 - Are everything initialized or reset properly?
 - 25 - Are you sure about the STL thing you are using?
 - Read `cppreference` (should be available).
 - 27 - Print everything and run it on pen and paper.
- 29 - Time Limit Exceeded:
 - Calculate your time complexity again.
 - 31 - Does the program actually end?
 - Check for `'while(q.size())'` etc.
 - 33 - Test the largest cases locally.
 - 35 - Did you do unnecessary stuff?
 - e.g. pass vectors by value
 - e.g. `'memset'` for every test case
 - 37 - Is your constant factor reasonable?
- 39 - Runtime Error:
 - 41 - Check memory usage.
 - Forget to clear or destroy stuff?
 - > `'vector::shrink_to_fit()'`
 - 43 - Stack overflow?
 - Bad pointer / array access?
 - Try `'-fsanitize=address'`
 - 45 - Division by zero? NaN's?