

Contents

1 Misc	2	4.2.1 Matroid Intersection	16
1.1 Contest	2	4.2.2 De Bruijn Sequence	16
1.1.1 Makefile	2	4.2.3 Multinomial	17
1.2 How Did We Get Here?	2	4.3 Algebra	17
1.2.1 Macros	2	4.3.1 Formal Power Series	17
1.2.2 Fast I/O	2	4.4 Theorems	18
1.2.3 constexpr	2	4.4.1 Kirchhoff's Theorem	18
1.2.4 Bump Allocator	2	4.4.2 Tutte's Matrix	18
1.3 Tools	3	4.4.3 Cayley's Formula	18
1.3.1 Floating Point Binary Search	3	4.4.4 Erdős–Gallai Theorem	18
1.3.2 SplitMix64	3	4.4.5 Burnside's Lemma	18
1.3.3 <random>	3		
1.3.4 x86 Stack Hack	3		
1.4 Algorithms	3	5 Numeric	19
1.4.1 Bit Hacks	3	5.1 Barrett Reduction	19
1.4.2 Aliens Trick	3	5.2 Long Long Multiplication	19
1.4.3 Hilbert Curve	3	5.3 Fast Fourier Transform	19
1.4.4 Infinite Grid Knight Distance	3	5.4 Fast Walsh-Hadamard Transform	19
1.4.5 Poker Hand	3	5.5 Subset Convolution	19
1.4.6 Longest Increasing Subsequence	3	5.6 Linear Recurrences	19
1.4.7 Mo's Algorithm on Tree	4	5.6.1 Berlekamp-Massey Algorithm	19
2 Data Structures	5	5.6.2 Linear Recurrence Calculation	19
2.1 GNU PBDS	5	5.7 Matrices	20
2.2 Segment Tree (ZKW)	5	5.7.1 Determinant	20
2.3 Line Container	5	5.7.2 Inverse	20
2.4 Li-Chao Tree	5	5.7.3 Characteristic Polynomial	20
2.5 Heavy-Light Decomposition	5	5.7.4 Solve Linear Equation	21
2.6 Wavelet Matrix	6	5.8 Polynomial Interpolation	21
2.7 Link-Cut Tree	6	5.9 Simplex Algorithm	21
3 Graph	8		
3.1 Modeling	8	6 Geometry	23
3.2 Matching/Flows	8	6.1 Point	23
3.2.1 Dinic's Algorithm	8	6.1.1 Quaternion	23
3.2.2 Minimum Cost Flow	8	6.1.2 Spherical Coordinates	23
3.2.3 Gomory-Hu Tree	9	6.2 Segments	23
3.2.4 Global Minimum Cut	9	6.3 Convex Hull	23
3.2.5 Bipartite Minimum Cover	9	6.3.1 3D Hull	23
3.2.6 Edmonds' Algorithm	9	6.4 Angular Sort	24
3.2.7 Minimum Weight Matching	10	6.5 Convex Polygon Minkowski Sum	24
3.2.8 Stable Marriage	10	6.6 Point In Polygon	24
3.2.9 Kuhn-Munkres algorithm	11	6.6.1 Convex Version	24
3.3 Shortest Path Faster Algorithm	11	6.6.2 Offline Multiple Points Version	24
3.4 Strongly Connected Components	11	6.7 Closest Pair	25
3.4.1 2-Satisfiability	12	6.8 Minimum Enclosing Circle	25
3.5 Biconnected Components	12	6.9 Delaunay Triangulation	25
3.5.1 Articulation Points	12	6.9.1 Slower Version	26
3.5.2 Bridges	12	6.10 Half Plane Intersection	26
3.6 Triconnected Components	12		
3.7 Centroid Decomposition	12		
3.8 Minimum Mean Cycle	13		
3.9 Directed MST	13		
3.10 Maximum Clique	13		
3.11 Dominator Tree	14		
3.12 Manhattan Distance MST	14		
4 Math	15		
4.1 Number Theory	15		
4.1.1 Mod Struct	15		
4.1.2 Miller-Rabin	15		
4.1.3 Linear Sieve	15		
4.1.4 Get Factors	15		
4.1.5 Binary GCD	15		
4.1.6 Extended GCD	15		
4.1.7 Chinese Remainder Theorem	15		
4.1.8 Baby-Step Giant-Step	15		
4.1.9 Pollard's Rho	16		
4.1.10 Tonelli-Shanks Algorithm	16		
4.1.11 Chinese Sieve	16		
4.1.12 Rational Number Binary Search	16		
4.1.13 Farey Sequence	16		
4.2 Combinatorics	16		

1. Misc

1.1. Contest

1.1.1. Makefile

```

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

```

1.2. How Did We Get Here?

1.2.1. Macros

Use vectorizations and math optimizations at your own peril.
For gcc \geq 9, there are `[[likely]]` and `[[unlikely]]` attributes.
Call gcc with `-fopt-info-optimized-missed-optall` for optimization info.

```

1 #define _GLIBCXX_DEBUG           1 // for debug mode
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
# pragma GCC ivdep

```

1.2.2. Fast I/O

```

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

```

```

59     } write_(++tbuf);
60 }

```

Kotlin

```

1 import java.io.*
2 import java.util.*
3
4 @JvmField val cin = System.`in`.bufferedReader()
5 @JvmField val cout = PrintWriter(System.out, false)
6 @JvmField var tokenizer: StringTokenizer = StringTokenizer("")
7 fun nextLine() = cin.readLine()!!
8 fun read(): String {
9     while (!tokenizer.hasMoreTokens())
10        tokenizer = StringTokenizer(nextLine())
11    return tokenizer.nextToken()
12}
13
14// example
15fun main() {
16    val n = read().toInt()
17    val a = DoubleArray(n) { read().toDouble() }
18    cout.println("omg hi")
19    cout.flush()
20}

```

1.2.3. constexpr

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

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

```

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

```

1.2.4. Bump Allocator

```

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

```

1.3. Tools

1.3.1. Floating Point Binary Search

```

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

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);
}
```

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
// random uint_fast64_t: RNG();
// uniform random of type T(int, double, ...) in [l, r];
// uniform_int_distribution<T> dist(l, r); dist(RNG);
11 }
```

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 }
// iterate over all (proper) subsets of bitset s
6 void subsets(ull s) {
7     for (ull x = s; x;) { --x &= s; /* do stuff */ }
8 }
```

1.4.2. Aliens Trick

```

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

1.4.3. Hilbert Curve

```

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

1.4.4. Infinite Grid Knight Distance

```

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

1.4.5. Poker Hand

```

1
2
3
4
5
6
7 using namespace std;
8
9 struct hand {
10     static constexpr auto rk = [] {
11         array<int, 256> x{};
12         auto s = "23456789TJQKACDH$";
13         for (int i = 0; i < 17; i++) x[s[i]] = i % 13;
14         return x;
15     }();
16     vector<pair<int, int>> v;
17     vector<int> cnt, vf, vs;
18     int type;
19     hand() : cnt(4), type(0) {}
20     void add_card(char suit, char rank) {
21         ++cnt[rk[suit]];
22         for (auto &[f, s] : v)
23             if (s == rk[rank]) return ++f, void();
24         v.emplace_back(1, rk[rank]);
25     }
26     void process() {
27         sort(v.rbegin(), v.rend());
28         for (auto [f, s] : v) vf.push_back(f), vs.push_back(s);
29         bool str = 0, flu = find(all(cnt), 5) != cnt.end();
30         if ((str = v.size() == 5))
31             for (int i = 1; i < 5; i++)
32                 if (vs[i] != vs[i - 1] + 1) str = 0;
33         if (vs == vector<int>{12, 3, 2, 1, 0})
34             str = 1, vs = {3, 2, 1, 0, -1};
35         if (str && flu) type = 9;
36         else if (vf[0] == 4) type = 8;
37         else if (vf[0] == 3 && vf[1] == 2) type = 7;
38         else if (str || flu) type = 5 + flu;
39         else if (vf[0] == 3) type = 4;
40         else if (vf[0] == 2) type = 2 + (vf[1] == 2);
41         else type = 1;
42     }
43     bool operator<(const hand &b) const {
44         return make_tuple(type, vf, vs) <
45                make_tuple(b.type, b.vf, b.vs);
46     }
47 }
```

1.4.6. Longest Increasing Subsequence

```

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

```

15    }
16    int L = sz(res), cur = res.back().second;
17    vi ans(L);
18    while (L--) ans[L] = cur, cur = prev[cur];
19    return ans;
}

```

1.4.7. Mo's Algorithm on Tree

```

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

```

2. Data Structures

2.1. GNU PBDS

2.2. Segment Tree (ZKW)

```

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

```

2.3. Line Container

```

1
2
3 struct Line {
4     mutable ll k, m, p;
5     bool operator<(const Line &o) const { return k < o.k; }
6     bool operator<(ll x) const { return p < x; }
7 };
8 // add: line y=kx+m, query: maximum y of given x
9 struct LineContainer : multiset<Line, less<> {
10    // (for doubles, use inf = 1/.0, div(a,b) = a/b)
11    static const ll inf = LLONG_MAX;
12    ll div(ll a, ll b) { // floored division
13        return a / b - ((a ^ b) < 0 && a % b);
14    }
15    bool isect(iterator x, iterator y) {
16        if (y == end()) return x->p = inf, 0;
17        if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
18        else x->p = div(y->m - x->m, x->k - y->k);
19        return x->p >= y->p;
20    }
21    void add(ll k, ll m) {
22        auto z = insert({k, m, 0}), y = z++, x = y;
23        while (isect(y, z)) z = erase(z);
24        if (x != begin() && isect(--x, y))
25            isect(x, y = erase(y));
26        while ((y = x) != begin() && (--x)->p >= y->p)
27            isect(x, erase(y));
28    }
29    ll query(ll x) {
30        assert(!empty());
31        auto l = *lower_bound(x);
32        return l.k * x + l.m;
33    }
34};

```

2.4. Li-Chao Tree

```

1 constexpr ll MAXN = 2e5, INF = 2e18;
2 struct Line {
3     ll m, b;
4     Line() : m(0), b(-INF) {}
5     Line(ll _m, ll _b) : m(_m), b(_b) {}
6     ll operator()(ll x) const { return m * x + b; }
7 };
8 struct Li_Chao {
9     Line a[MAXN * 4];
10    void insert(Line seg, int l, int r, int v = 1) {
11        if (l == r) {
12            if (seg(l) > a[v](l)) a[v] = seg;
13            return;
14        }
15        int mid = (l + r) >> 1;
16        if (a[v].m > seg.m) swap(a[v], seg);
17        if (a[v](mid) < seg(mid)) {
18            swap(a[v], seg);
19            insert(seg, l, mid, v << 1);
20        } else insert(seg, mid + 1, r, v << 1 | 1);
21    }
22    ll query(int x, int l, int r, int v = 1) {
23        if (l == r) return a[v](x);
24        int mid = (l + r) >> 1;
25        if (x <= mid)
26            return max(a[v](x), query(x, l, mid, v << 1));
27        else
28            return max(a[v](x), query(x, mid + 1, r, v << 1 | 1));
29    }
30};

```

2.5. Heavy-Light Decomposition

2.6. Wavelet Matrix

```

1
3 #pragma GCC target("popcnt,bmi2")
4 #include <immintrin.h>
5
6 // T is unsigned. You might want to compress values first
7 template <typename T> struct wavelet_matrix {
8     static_assert(is_unsigned_v<T>, "only unsigned T");
9     struct bit_vector {
10         static constexpr uint W = 64;
11         uint n, cnt0;
12         vector<ull> bits;
13         vector<uint> sum;
14         bit_vector(uint n_) : n(n_), bits(n / W + 1), sum(n / W + 1) {}
15         void build() {
16             for (uint j = 0; j != n / W; ++j)
17                 sum[j + 1] = sum[j] + _mm_popcnt_u64(bits[j]);
18             cnt0 = rank0(n);
19         }
20         void set_bit(uint i) { bits[i / W] |= 1ULL << i % W; }
21         bool operator[](uint i) const {
22             return !(bits[i / W] & 1ULL << i % W);
23         }
24         uint rank1(uint i) const {
25             return sum[i / W] +
26                 _mm_popcnt_u64(_bzhi_u64(bits[i / W], i % W));
27         }
28         uint rank0(uint i) const { return i - rank1(i); }
29     };
30     uint n, lg;
31     vector<bit_vector> b;
32     wavelet_matrix(const vector<T> &a) : n(a.size()) {
33         lg =
34             __lg(max(*max_element(a.begin(), a.end()), T(1))) + 1;
35         b.assign(lg, n);
36         vector<T> cur = a, nxt(n);
37         for (int h = lg; h--;) {
38             for (uint i = 0; i < n; ++i)
39                 if (cur[i] & (T(1) << h)) b[h].set_bit(i);
40             b[h].build();
41             int il = 0, ir = b[h].cnt0;
42             for (uint i = 0; i < n; ++i)
43                 nxt[(b[h][i] ? ir : il)++] = cur[i];
44             swap(cur, nxt);
45         }
46     }
47     T operator[](uint i) const {
48         T res = 0;
49         for (int h = lg; h--;) {
50             if (b[h][i])
51                 i += b[h].cnt0 - b[h].rank0(i), res |= T(1) << h;
52             else i = b[h].rank0(i);
53         }
54         return res;
55     }
56     // query k-th smallest (0-based) in a[l, r)
57     T kth(uint l, uint r, uint k) const {
58         T res = 0;
59         for (int h = lg; h--;) {
60             uint tl = b[h].rank0(l), tr = b[h].rank0(r);
61             if (k >= tr - tl) {
62                 k -= tr - tl;
63                 l += b[h].cnt0 - tl;
64                 r += b[h].cnt0 - tr;
65                 res |= T(1) << h;
66             } else l = tl, r = tr;
67         }
68         return res;
69     }
70     // count of i in [l, r) with a[i] < u
71     uint count(uint l, uint r, T u) const {
72         if (u >= T(1) << lg) return r - l;
73         uint res = 0;
74         for (int h = lg; h--;) {
75             uint tl = b[h].rank0(l), tr = b[h].rank0(r);
76             if (u & (T(1) << h)) {
77                 l += b[h].cnt0 - tl;
78                 r += b[h].cnt0 - tr;
79                 res += tr - tl;
80             } else l = tl, r = tr;
81         }
82         return res;
83     }
84 };

```

```

3 const int MXN = 100005;
4 const int MEM = 100005;
5
6 struct Splay {
7     static Splay nil, mem[MEM], *pmem;
8     Splay *ch[2], *f;
9     int val, rev, size;
10    Splay() : val(-1), rev(0), size(0) {
11        f = ch[0] = ch[1] = &nil;
12    }
13    Splay(int val) : val(val), rev(0), size(1) {
14        f = ch[0] = ch[1] = &nil;
15    }
16    bool isr() {
17        return f->ch[0] != this && f->ch[1] != this;
18    }
19    int dir() { return f->ch[0] == this ? 0 : 1; }
20    void setCh(Splay *c, int d) {
21        ch[d] = c;
22        if (c != &nil) c->f = this;
23        pull();
24    }
25    void push() {
26        if (rev) {
27            swap(ch[0], ch[1]);
28            if (ch[0] != &nil) ch[0]->rev ^= 1;
29            if (ch[1] != &nil) ch[1]->rev ^= 1;
30            rev = 0;
31        }
32    }
33    void pull() {
34        size = ch[0]->size + ch[1]->size + 1;
35        if (ch[0] != &nil) ch[0]->f = this;
36        if (ch[1] != &nil) ch[1]->f = this;
37    }
38 } Splay::nil, Splay::mem[MEM], *Splay::pmem = Splay::mem;
39 Splay *nil = &Splay::nil;
40
41 void rotate(Splay *x) {
42     Splay *p = x->f;
43     int d = x->dir();
44     if (!p->isr()) p->f->setCh(x, p->dir());
45     else 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()) rotate(x);
63         else if (x->dir() == x->f->dir())
64             rotate(x->f), rotate(x);
65         else rotate(x), rotate(x);
66     }
67 }
68
69 Splay *access(Splay *x) {
70     Splay *q = nil;
71     for (; x != nil; x = x->f) {
72         splay(x);
73         x->setCh(q, 1);
74         q = x;
75     }
76     return q;
77 }
78 void evert(Splay *x) {
79     access(x);
80     splay(x);
81     x->rev ^= 1;
82     x->push();
83     x->pull();
84 }
85 void link(Splay *x, Splay *y) {
86     // evert(x);
87     access(x);
88     splay(x);
89     evert(y);
90     x->setCh(y, 1);
91 }
92 void cut(Splay *x, Splay *y) {
93     // evert(x);

```

2.7. Link-Cut Tree

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

3. Graph

3.1. Modeling

- Maximum/Minimum flow with lower bound / Circulation problem

1. Construct super source S and sink T .

2. For each edge (x, y, l, u) , connect $x \rightarrow y$ with capacity $u - l$.

3. For each vertex v , denote by $in(v)$ the difference between the sum of incoming lower bounds and the sum of outgoing lower bounds.

4. If $in(v) > 0$, connect $S \rightarrow v$ with capacity $in(v)$, otherwise, connect $v \rightarrow T$ with capacity $-in(v)$.

– To maximize, connect $t \rightarrow s$ with capacity ∞ (skip this in circulation problem), and let f be the maximum flow from S to T .

If $f \neq \sum_{v \in V, in(v) > 0} in(v)$, there's no solution. Otherwise, the maximum flow from s to t is the answer.

– To minimize, let f be the maximum flow from S to T . Connect $t \rightarrow s$ with capacity ∞ and let the flow from S to T be f' . If $f + f' \neq \sum_{v \in V, in(v) > 0} in(v)$, there's no solution. Otherwise, f'

is the answer.

5. The solution of each edge e is $l_e + f_e$, where f_e corresponds to the flow of edge e on the graph.

- Construct minimum vertex cover from maximum matching M on bipartite graph (X, Y)

1. Redirect every edge: $y \rightarrow x$ if $(x, y) \in M$, $x \rightarrow y$ otherwise.

2. DFS from unmatched vertices in X .

3. $x \in X$ is chosen iff x is unvisited.

4. $y \in Y$ is chosen iff y is visited.

- Minimum cost cyclic flow

1. Construct super source S and sink T

2. For each edge (x, y, c) , connect $x \rightarrow y$ with $(cost, cap) = (c, 1)$ if $c > 0$, otherwise connect $y \rightarrow x$ with $(cost, cap) = (-c, 1)$

3. For each edge with $c < 0$, sum these cost as K , then increase $d(y)$ by 1, decrease $d(x)$ by 1

4. For each vertex v with $d(v) > 0$, connect $S \rightarrow v$ with $(cost, cap) = (0, d(v))$

5. For each vertex v with $d(v) < 0$, connect $v \rightarrow T$ with $(cost, cap) = (0, -d(v))$

6. Flow from S to T , the answer is the cost of the flow $C + K$

- Maximum density induced subgraph

1. Binary search on answer, suppose we're checking answer T

2. Construct a max flow model, let K be the sum of all weights

3. Connect source $s \rightarrow v$, $v \in G$ with capacity K

4. For each edge (u, v, w) in G , connect $u \rightarrow v$ and $v \rightarrow u$ with capacity w

5. For $v \in G$, connect it with sink $v \rightarrow t$ with capacity $K + 2T - (\sum_{e \in E(v)} w(e)) - 2w(v)$

6. T is a valid answer if the maximum flow $f < K|V|$

- Minimum weight edge cover

1. For each $v \in V$ create a copy v' , and connect $u' \rightarrow v'$ with weight $w(u, v)$.

2. Connect $v \rightarrow v'$ with weight $2\mu(v)$, where $\mu(v)$ is the cost of the cheapest edge incident to v .

3. Find the minimum weight perfect matching on G' .

- Project selection problem

1. If $p_v > 0$, create edge (s, v) with capacity p_v ; otherwise, create edge (v, t) with capacity $-p_v$.

2. Create edge (u, v) with capacity w with w being the cost of choosing u without choosing v .

3. The mincut is equivalent to the maximum profit of a subset of projects.

- 0/1 quadratic programming

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

can be minimized by the mincut of the following graph:

1. Create edge (x, t) with capacity c_x and create edge (s, y) with capacity c_y .

2. Create edge (x, y) with capacity c_{xy} .

3. Create edge (x, y) and edge (x', y') with capacity $c_{xyx'y'}$.

3.2. Matching/Flows

3.2.1. Dinic's Algorithm

```

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

```

3.2.2. Minimum Cost Flow

```

1 struct MCF {
2     struct edge {
3         ll to, from, cap, flow, cost, rev;
4     } * fromE[MAXN];
5     vector<edge> v[MAXN];
6     ll n, s, t, flows[MAXN], dis[MAXN], pi[MAXN], flowlim;
7     void make_edge(int s, int t, ll cap, ll cost) {
8         if (!cap) return;
9         v[s].pb(edge{t, s, cap, 0LL, cost, v[t].size()});
10        v[t].pb(edge{s, t, 0LL, 0LL, -cost, v[s].size() - 1});
11    }
12    bitset<MAXN> vis;
13    void dijkstra() {
14        vis.reset();
15        gnu_pbds::priority_queue<pair<ll, int>> q;
16        vector<decltype(q)::point_iterator> its(n);
17        q.push({0LL, s});
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});
32                    else q.modify(its[e.to], {-dis[e.to], e.to});
33                }
34            }
35        }
36    }
37};

```

```

33     }
35   }
36 }
37 bool AP(ll &flow) {
38   fill_n(dis, n, INF);
39   fromE[s] = 0;
40   dis[s] = 0;
41   flows[s] = flowlim - flow;
42   dijkstra();
43   if (dis[t] == INF) return false;
44   flow += flows[t];
45   for (edge *e = fromE[t]; e; e = fromE[e->from]) {
46     e->flow += flows[t];
47     v[e->to][e->rev].flow -= flows[t];
48   }
49   for (int i = 0; i < n; i++)
50     pi[i] = min(pi[i] + dis[i], INF);
51   return true;
52 }
53 pll solve(int _s, int _t, ll _flowlim = INF) {
54   s = _s, t = _t, flowlim = _flowlim;
55   pll re;
56   while (re.F != flowlim && AP(re.F))
57   ;
58   for (int i = 0; i < n; i++)
59     for (edge &e : v[i])
60       if (e.flow != 0) re.S += e.flow * e.cost;
61   re.S /= 2;
62   return re;
63 }
64 void init(int _n) {
65   n = _n;
66   fill_n(pi, n, 0);
67   for (int i = 0; i < n; i++) v[i].clear();
68 }
69 void setpi(int s) {
70   fill_n(pi, n, INF);
71   pi[s] = 0;
72   for (ll it = 0, flag = 1, tdis; flag && it < n; it++) {
73     flag = 0;
74     for (int i = 0; i < n; i++)
75       if (pi[i] != INF)
76         for (edge &e : v[i])
77           if (e.cap && (tdis = pi[i] + e.cost) < pi[e.to])
78             pi[e.to] = tdis, flag = 1;
79   }
80 }
81 };

```

3.2.3. Gomory-Hu Tree

Requires: Dinic's Algorithm

```

1
2
3 int e[MAXN][MAXN];
4 int p[MAXN];
5 Dinic D; // original graph
6 void gomory_hu() {
7   fill(p, p + n, 0);
8   fill(e[0], e[n], INF);
9   for (int s = 1; s < n; s++) {
10     int t = p[s];
11     Dinic F = D;
12     int tmp = F.max_flow(s, t);
13     for (int i = 1; i < s; i++)
14       e[s][i] = e[i][s] = min(tmp, e[t][i]);
15     for (int i = s + 1; i <= n; i++)
16       if (p[i] == t && F.side[i]) p[i] = s;
17   }
}

```

3.2.4. Global Minimum Cut

```

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

```

```

17   if (i == phase - 1) {
18     rep(j, 0, N) weights[prev][j] += weights[k][j];
19     rep(j, 0, N) weights[j][prev] = weights[prev][j];
20     used[k] = true;
21     cut.push_back(k);
22     if (best_weight == -1 || w[k] < best_weight) {
23       best_cut = cut;
24       best_weight = w[k];
25     }
26   } else {
27     rep(j, 0, N) w[j] += weights[k][j];
28     added[k] = true;
29   }
30 }
31 return {best_weight, best_cut};
32 }
33 }

```

3.2.5. Bipartite Minimum Cover

Requires: Dinic's Algorithm

```

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

```

3.2.6. Edmonds' Algorithm

```

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

```

```

19     }
20     vector<int> toJoin;
21     void mark_path(int v, int x, int b, vector<int> &path) {
22         for (; getBase(v) != b; v = p[x]) {
23             p[v] = x, x = pa[v];
24             toJoin.push_back(v);
25             toJoin.push_back(x);
26             if (!used[x]) used[x] = ++T, path.push_back(x);
27         }
28     }
29     bool go(int v) {
30         for (int x : g[v]) {
31             int b, bv = getBase(v), bx = getBase(x);
32             if (bv == bx) {
33                 continue;
34             } else if (used[x]) {
35                 vector<int> path;
36                 toJoin.clear();
37                 if (used[bx] < used[bv])
38                     mark_path(v, x, b = bx, path);
39                 else mark_path(x, v, b = bv, path);
40                 for (int z : toJoin) base[getBase(z)] = b;
41                 for (int z : path)
42                     if (go(z)) return 1;
43             } else if (p[x] == -1) {
44                 p[x] = v;
45                 if (pa[x] == -1) {
46                     for (int y; x != -1; x = v)
47                         y = p[x], v = pa[y], pa[x] = y, pa[y] = x;
48                     return 1;
49                 }
50                 if (!used[pa[x]]) {
51                     used[pa[x]] = ++T;
52                     if (go(pa[x])) return 1;
53                 }
54             }
55         }
56         return 0;
57     }
58     void init_dfs() {
59         for (int i = 0; i < n; i++)
60             used[i] = 0, p[i] = -1, base[i] = i;
61     }
62     bool dfs(int root) {
63         used[root] = ++T;
64         return go(root);
65     }
66     void match() {
67         int ans = 0;
68         for (int v = 0; v < n; v++)
69             for (int x : g[v])
70                 if (pa[v] == -1 && pa[x] == -1) {
71                     pa[v] = x, pa[x] = v, ans++;
72                     break;
73                 }
74         init_dfs();
75         for (int i = 0; i < n; i++)
76             if (pa[i] == -1 && dfs(i)) ans++, init_dfs();
77         cout << ans * 2 << "\n";
78         for (int i = 0; i < n; i++)
79             if (pa[i] > i)
80                 cout << i + 1 << " " << pa[i] + 1 << "\n";
81     }

```

3.2.7. Minimum Weight Matching

```

1 struct Graph {
2     static const int MAXN = 105;
3     int n, e[MAXN][MAXN];
4     int match[MAXN], d[MAXN], onstk[MAXN];
5     vector<int> stk;
6     void init(int _n) {
7         n = _n;
8         for (int i = 0; i < n; i++)
9             for (int j = 0; j < n; j++)
10                 // change to appropriate infinity
11                 // if not complete graph
12                 e[i][j] = 0;
13     }
14     void add_edge(int u, int v, int w) {
15         e[u][v] = e[v][u] = w;
16     }
17     bool SPFA(int u) {
18         if (onstk[u]) return true;
19         stk.push_back(u);
20         onstk[u] = 1;
21         for (int v = 0; v < n; v++) {
22             if (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);
28                     if (SPFA(m)) return true;
29                     stk.pop_back();
30                     onstk[v] = 0;
31                 }
32             }
33             onstk[u] = 0;
34             stk.pop_back();
35             return false;
36         }
37         int solve() {
38             for (int i = 0; i < n; i += 2) {
39                 match[i] = i + 1;
40                 match[i + 1] = i;
41             }
42             while (true) {
43                 int found = 0;
44                 for (int i = 0; i < n; i++) onstk[i] = d[i] = 0;
45                 for (int i = 0; i < n; i++) {
46                     stk.clear();
47                     if (!onstk[i] && SPFA(i)) {
48                         found = 1;
49                         while (stk.size() >= 2) {
50                             int u = stk.back();
51                             stk.pop_back();
52                             int v = stk.back();
53                             stk.pop_back();
54                             match[u] = v;
55                             match[v] = u;
56                         }
57                     }
58                     if (!found) break;
59                 }
60                 int ret = 0;
61                 for (int i = 0; i < n; i++) ret += e[i][match[i]];
62                 ret /= 2;
63                 return ret;
64             }
65         } graph;
66     }

```

3.2.8. 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
36     queue<int> que;
37     for (int i = 0; i < n; i++) que.push(i);
38     while (!que.empty()) {
39         int boy_id = que.front();
40         que.pop();
41
42         int girl_id = favor[boy_id][current[boy_id]];
43         current[boy_id]++;
44
45     }

```

```

47     if (order[girl_id][boy_id] <
48         order[girl_id][girl_current[girl_id]]) {
49         if (girl_current[girl_id] < n)
50             que.push(girl_current[girl_id]);
51         girl_current[girl_id] = boy_id;
52     } else {
53         que.push(boy_id);
54     }
55 }

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

```

3.2.9. 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] =
24                    min(slack[y], lx[x] + ly[y] - edge[x][y]);
25            } else {
26                vy[y] = 1;
27                if (match[y] == -1 || DFS(match[y])) {
28                    match[y] = x;
29                    return true;
30                }
31            }
32        }
33        return false;
34    }
35    ll solve() {
36        fill(match, match + n, -1);
37        fill(lx, lx + n, -INF);
38        fill(ly, ly + n, 0);
39        for (int i = 0; i < n; i++)
40            for (int j = 0; j < n; j++)
41                lx[i] = max(lx[i], edge[i][j]);
42        for (int i = 0; i < n; i++) {

```

```

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

```

3.3. Shortest Path Faster Algorithm

```

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

```

3.4. Strongly Connected Components

```

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

```

```

21   for (int y = -1; y != x;) {
22     y = stk.back();
23     stk.pop_back();
24     instk[y] = 0;
25     scc.back().push_back(y);
26   }
27 }
28 void solve() {
29   for (int i = 0; i < n; i++) {
30     if (!time[i]) dfs(i);
31   reverse(scc.begin(), scc.end());
32   // scc in topological order
33 }
34 };
35 };

```

3.4.1. 2-Satisfiability

Requires: Strongly Connected Components

```

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

```

3.5. Biconnected Components

3.5.1. Articulation Points

```

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

```

3.5.2. Bridges

```

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

```

```

15   }
16   if (tin[x] == low[x]) {
17     ++sz;
18     while (st.size()) {
19       int u = st.top();
20       st.pop();
21       bcc[u] = sz;
22       if (u == x) break;
23     }
24   }
25 }

```

3.6. Triconnected Components

```

1
2 // requires a union-find data structure
3 struct ThreeEdgeCC {
4   int V, ind;
5   vector<int> id, pre, post, low, deg, path;
6   vector<vector<int>> components;
7   UnionFind uf;
8   template <class Graph>
9   void dfs(const Graph &G, int v, int prev) {
10     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         }
32         continue;
33       }
34     dfs(G, w, v);
35     if (path[w] == -1 && deg[w] <= 1) {
36       deg[v] += deg[w];
37       low[v] = min(low[v], low[w]);
38       continue;
39     }
40     if (deg[w] == 0) w = path[w];
41     if (low[v] > low[w]) {
42       low[v] = min(low[v], low[w]);
43       swap(w, path[v]);
44     }
45     for (; w != -1; w = path[w]) {
46       uf.join(v, w);
47       deg[v] += deg[w];
48     }
49   }
50   post[v] = ind;
51 }
52 template <class Graph>
53 ThreeEdgeCC(const Graph &G)
54   : V(G.size()), ind(-1), id(V, -1), pre(V, -1),
55   post(V, low(V, INT_MAX), deg(V, 0), path(V, -1),
56   uf(V) {
57   for (int v = 0; v < V; v++)
58     if (pre[v] == -1) dfs(G, v, -1);
59   components.reserve(uf.cnt);
60   for (int v = 0; v < V; v++)
61     if (uf.find(v) == v) {
62       id[v] = components.size();
63       components.emplace_back(1, v);
64       components.back().reserve(uf.getSize(v));
65     }
66   for (int v = 0; v < V; v++)
67     if (id[v] == -1)
68       components[id[v]] = id[uf.find(v)].push_back(v);
69   }
70 }
71

```

3.7. Centroid Decomposition

```

1 void get_center(int now) {
2   v[now] = true;
3   vtx.push_back(now);

```

```

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

```

3.8. Minimum Mean Cycle

```

1
2 // d[i][j] == 0 if {i,j} !in E
3 long long d[1003][1003], dp[1003][1003];
4
5 pair<long long, long long> MMWC() {
6   memset(dp, 0x3f, sizeof(dp));
7   for (int i = 1; i <= n; ++i) dp[0][i] = 0;
8   for (int i = 1; i <= n; ++i) {
9     for (int j = 1; j <= n; ++j) {
10       for (int k = 1; k <= n; ++k) {
11         dp[i][k] = min(dp[i - 1][j] + d[j][k], dp[i][k]);
12       }
13     }
14   }
15   long long au = 1ll << 31, ad = 1;
16   for (int i = 1; i <= n; ++i) {
17     if (dp[n][i] == 0x3f3f3f3f3f3f3f) continue;
18     long long u = 0, d = 1;
19     for (int j = n - 1; j >= 0; --j) {
20       if ((dp[n][i] - dp[j][i]) * d > u * (n - j)) {
21         u = dp[n][i] - dp[j][i];
22         d = n - j;
23       }
24     }
25     if (u * ad < au * d) au = u, ad = d;
26   }
27   long long g = __gcd(au, ad);
28   return make_pair(au / g, ad / g);
29 }

```

3.9. Directed MST

```

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

```

```

21        if (!inc[i]) {
22          for (int j = 1; j <= n; ++j) {
23            if (!inc[j] && i != j && g[j][i] < fw[i]) {
24              fw[i] = g[j][i];
25              fr[i] = j;
26            }
27          }
28        }
29        int x = -1;
30        for (int i = 1; i <= n; ++i)
31          if (i != root && !inc[i]) {
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) continue;
36            else {
37              x = i;
38              break;
39            }
40          }
41        if (!~x) {
42          for (int i = 1; i <= n; ++i)
43            if (i != root && !inc[i]) ans += fw[i];
44          return ans;
45        }
46        int y = x;
47        for (int i = 1; i <= n; ++i) vis[i] = false;
48        do {
49          ans += fw[y];
50          y = fr[y];
51          vis[y] = inc[y] = true;
52        } while (y != x);
53        inc[x] = false;
54        for (int k = 1; k <= n; ++k)
55          if (vis[k]) {
56            for (int j = 1; j <= n; ++j)
57              if (!vis[j]) {
58                if (g[x][j] > g[k][j]) g[x][j] = g[k][j];
59                if (g[j][k] < inf &&
60                    g[j][k] - fw[k] < g[j][x])
61                  g[j][x] = g[j][k] - fw[k];
62              }
63            }
64        }
65      }
66    }
67    int dfs(int now) {
68      int r = 1;
69      vis[now] = true;
70      for (int i = 1; i <= n; ++i)
71        if (g[now][i] < inf && !vis[i]) r += dfs(i);
72      return r;
73    }
74 }

```

3.10. Maximum Clique

```

1 // source: KACTL
2
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    }
29    vv T;
30    for (auto v : R)
31      if (e[R.back().i][v.i]) T.push_back({v.i});
32    if (sz(T)) {
33      if (S[lev]++ / ++pk < limit) init(T);
34      int j = 0, mxk = 1,
35      mnk = max(sz(qmax) - sz(q) + 1, 1);
36    }
37  }

```

```

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

```

```

63         if (cmp(sdom[mn[v]], sdom[u]))
64             sdom[u] = sdom[mn[v]];
65     }
66     cov[sdom[u]].push_back(u);
67     mom[u] = par[u];
68     for (int w : cov[par[u]]) {
69         eval(w);
70         if (cmp(sdom[mn[w]], par[u])) idom[w] = mn[w];
71         else idom[w] = par[u];
72     }
73     cov[par[u]].clear();
74 }
75 REP1(i, 2, ts) {
76     int u = nfd[i];
77     if (u == 0) continue;
78     if (idom[u] != sdom[u]) idom[u] = idom[idom[u]];
79 }
80 dom;

```

3.11. Dominator Tree

```

1 // idom[n] is the unique node that strictly dominates n but
2 // does not strictly dominate any other node that strictly
3 // dominates n. idom[n] = 0 if n is entry or the entry
4 // cannot reach n.
5 struct DominatorTree {
6     static const int MAXN = 200010;
7     int n, s;
8     vector<int> g[MAXN], pred[MAXN];
9     vector<int> cov[MAXN];
10    int dfn[MAXN], nfd[MAXN], ts;
11    int par[MAXN];
12    int sdom[MAXN], idom[MAXN];
13    int mn[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 (mn[u] == u) return u;
19        int res = eval(mn[u]);
20        if (cmp(sdom[mn[mn[u]]], sdom[mn[u]]))
21            mn[u] = mn[mn[u]];
22        return mn[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            mn[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                }
63        }
64    }
65}
```

3.12. Manhattan Distance MST

```

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

```

4. Math

4.1. Number Theory

4.1.1. Mod Struct

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

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

Requires: Extended GCD

```

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

```

4.1.2. Miller-Rabin

Requires: Mod Struct

```

1
3 // checks if Mod::MOD is prime
4 bool is_prime() {
5     if (MOD < 2 || MOD % 2 == 0) return MOD == 2;
6     Mod A[] = {2, 7, 61}; // for int values (< 2^31)
7     // ll: 2, 325, 9375, 28178, 450775, 9780504, 1795265022
8     int s = __builtin_ctzll(MOD - 1), i;
9     for (Mod a : A) {
10         Mod x = a ^ (MOD >> s);
11         for (i = 0; i < s && (x + 1).v > 2; i++) x *= x;
12         if (i && x != -1) return 0;
13     }
14     return 1;
15 }

```

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    }
18 }

```

```

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         else phi[i * p] = phi[i] * (p - 1);
25     }
26 }
27 }

```

4.1.4. Get Factors

Requires: Linear Sieve

```

1
3 vector<ll> all_factors(ll n) {
4     vector<ll> fac = {1};
5     while (n > 1) {
6         const ll p = mpf[n];
7         vector<ll> cur = {1};
8         while (n % p == 0) {
9             n /= p;
10            cur.push_back(cur.back() * p);
11        }
12        vector<ll> tmp;
13        for (auto x : fac)
14            for (auto y : cur) tmp.push_back(x * y);
15        tmp.swap(fac);
16    }
17    return fac;
18 }

```

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, b);
8         swap(s -= q * t, t, t);
9         swap(u -= q * v, v, v);
10    }
11    return {s, u, a};
12 }

```

4.1.7. Chinese Remainder Theorem

Requires: Extended GCD

```

1 // for 0 <= a < m, 0 <= b < n, returns the smallest x >= 0
2 // such that x % m == a and x % n == b
3 ll crt(ll a, ll m, ll b, ll n) {
4     if (n > m) swap(a, b), swap(m, n);
5     auto [x, y, g] = extgcd(m, n);
6     assert((a - b) % g == 0); // no solution
7     x = ((b - a) / g * x) % (n / g) * m + a;
8     return x < 0 ? x + m / g * n : x;
9 }

```

4.1.8. Baby-Step Giant-Step

Requires: Mod Struct

```

1
3 // returns x such that a ^ x = b where x \in [l, r)
4 ll bsgs(Mod a, Mod b, ll l = 0, ll r = MOD - 1) {
5     int m = sqrt(r - l) + 1, i;
6     unordered_map<ll, ll> tb;
7     Mod d = (a ^ l) / b;
8     for (i = 0, d = (a ^ l) / b; i < m; i++, d *= a)
9         if (d == 1) return l + i;
10        else tb[(ll)d] = l + i;
11    Mod c = Mod(1) / (a ^ m);
12    for (i = 0, d = 1; i < m; i++, d *= c)
13        if (auto j = tb.find((ll)d); j != tb.end())
14            return j->second + i * m;
15    return assert(0), -1; // no solution
16 }

```

4.1.9. Pollard's Rho

```

1 ll f(ll x, ll mod) { return (x * x + 1) % mod; }
// n should be composite
3 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

```

1
3 int legendre(Mod a) {
    if (a == 0) return 0;
    return (a ^ ((MOD - 1) / 2)) == 1 ? 1 : -1;
}
5 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

```

1 const ll N = 1000000;
// f, g, h multiplicative, h = f (dirichlet convolution) g
3 ll pre_g(ll n);
ll pre_h(ll n);
5 // preprocessed prefix sum of f
ll pre_f[N];
7 // 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

```

1 struct QQ {
    ll p, q;
3     QQ go(QQ b, ll d) { return {p + b.p * d, q + b.q * d}; }
};
5 bool pred(QQ);
// returns smallest p/q in [lo, hi] such that
7 // 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))

```

```

19         t++;
20         else len += step;
21         swap(lo, hi = hi.go(lo, len));
22         (dir ? L : H) = !len;
23     }
24     return dir ? hi : lo;
}

```

4.1.13. Farey Sequence

```

1 // returns (e/f), where (a/b, c/d, e/f) are
// three consecutive terms in the order n farey sequence
3 // to start, call next_farey(n, 0, 1, 1, n)
5 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.

```

1 constexpr int N = 100;
constexpr int INF = 1e9;
3
5 struct Matroid { // represents an independent set
    Matroid(bitset<N>); // initialize from an independent set
    bool can_add(int); // if adding will break independence
    Matroid remove(int); // removing from the set
};
7
9 auto matroid_intersection(int n, const vector<int> &w) {
11     bitset<N> S;
13     for (int sz = 1; sz <= n; sz++) {
14         Matroid M1(S), M2(S);
15
16         vector<vector<pii>> e(n + 2);
17         for (int j = 0; j < n; j++)
18             if (!S[j]) {
19                 if (M1.can_add(j)) e[n].emplace_back(j, -w[j]);
20                 if (M2.can_add(j)) e[j].emplace_back(n + 1, 0);
21             }
22         for (int i = 0; i < n; i++)
23             if (S[i]) {
24                 Matroid T1 = M1.remove(i), T2 = M2.remove(i);
25                 for (int j = 0; j < n; j++)
26                     if (!S[j]) {
27                         if (T1.can_add(j)) e[i].emplace_back(j, -w[j]);
28                         if (T2.can_add(j)) e[j].emplace_back(i, w[i]);
29                     }
30
31         vector<pii> dis(n + 2, {INF, 0});
32         vector<int> prev(n + 2, -1);
33         dis[0] = {0, 0};
34         // change to SPFA for more speed, if necessary
35         bool upd = 1;
36         while (upd) {
37             upd = 0;
38             for (int u = 0; u < n + 2; u++)
39                 for (auto [v, c] : e[u])
40                     pii x(dis[u].first + c, dis[u].second + 1);
41                     if (x < dis[v]) dis[v] = x, prev[v] = u, upd = 1;
42             }
43
44         if (dis[n + 1].first < INF)
45             for (int x = prev[n + 1]; x != n; x = prev[x])
46                 S.flip(x);
47             else break;
48
49         // S is the max-weighted independent set with size sz
50     }
51     return S;
52 }

```

4.2.2. De Bruijn Sequence

```

1 int res[kN], aux[kN], a[kN], sz;
void Rec(int t, int p, int n, int k) {
    if (t > n) {
        if (n % p == 0)
            for (int i = 1; i <= p; ++i) res[sz++] = aux[i];
        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;
}

```

4.2.3. Multinomial

```

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

```

4.3. Algebra

4.3.1. Formal Power Series

```

1
2
3 template <typename mint>
4 struct FormalPowerSeries : vector<mint> {
5     using vector<mint>::vector;
6     using FPS = FormalPowerSeries;
7
8     FPS &operator+=(const FPS &r) {
9         if (r.size() > this->size()) this->resize(r.size());
10        for (int i = 0; i < (int)r.size(); i++)
11            (*this)[i] += r[i];
12        return *this;
13    }
14
15    FPS &operator+=(const mint &r) {
16        if (this->empty()) this->resize(1);
17        (*this)[0] += r;
18        return *this;
19    }
20
21    FPS &operator-=(const FPS &r) {
22        if (r.size() > this->size()) this->resize(r.size());
23        for (int i = 0; i < (int)r.size(); i++)
24            (*this)[i] -= r[i];
25        return *this;
26    }
27
28    FPS &operator-=(const mint &r) {
29        if (this->empty()) this->resize(1);
30        (*this)[0] -= r;
31        return *this;
32    }
33
34    FPS &operator*=(const mint &v) {
35        for (int k = 0; k < (int)this->size(); k++)
36            (*this)[k] *= v;
37        return *this;
38    }
39
40    FPS &operator/=(const FPS &r) {
41        if (this->size() < r.size()) {
42            this->clear();
43            return *this;
44        }
45        int n = this->size() - r.size() + 1;
46        if ((int)r.size() <= 64) {
47            FPS f(*this), g(r);
48            g.shrink();
49            mint coeff = g.back().inverse();
50            for (auto &x : g) x *= coeff;
51            int deg = (int)f.size() - (int)g.size() + 1;
52            int gs = g.size();
53            FPS quo(deg);
54            for (int i = deg - 1; i >= 0; i--) {
55                quo[i] = f[i + gs - 1];
56                for (int j = 0; j < gs; j++)
57                    f[i + j] -= quo[i] * g[j];
58            }
59        }
60        *this = quo * coeff;

```

```

61         this->resize(n, mint(0));
62         return *this;
63     }
64     return *this = ((*this).rev().pre(n) * r.rev().inv(n))
65             .pre(n)
66             .rev();
67 }
68
69 FPS &operator%=(const FPS &r) {
70     *this -= *this / r * r;
71     shrink();
72     return *this;
73 }
74
75 FPS operator+(const FPS &r) const {
76     return FPS(*this) += r;
77 }
78 FPS operator+(const mint &v) const {
79     return FPS(*this) += v;
80 }
81 FPS operator-(const FPS &r) const {
82     return FPS(*this) -= r;
83 }
84 FPS operator-(const mint &v) const {
85     return FPS(*this) -= v;
86 }
87 FPS operator*(const FPS &r) const {
88     return FPS(*this) *= r;
89 }
90 FPS operator*(const mint &v) const {
91     return FPS(*this) *= v;
92 }
93 FPS operator/(const FPS &r) const {
94     return FPS(*this) /= r;
95 }
96 FPS operator%=(const FPS &r) const {
97     return FPS(*this) %= r;
98 }
99 FPS operator-() const {
100    FPS ret(this->size());
101    for (int i = 0; i < (int)this->size(); i++)
102        ret[i] = -(*this)[i];
103    return ret;
104 }
105
106 void shrink() {
107     while (this->size() && this->back() == mint(0))
108         this->pop_back();
109 }
110
111 FPS rev() const {
112     FPS ret(*this);
113     reverse(begin(ret), end(ret));
114     return ret;
115 }
116
117 FPS dot(FPS r) const {
118     FPS ret(min(this->size(), r.size()));
119     for (int i = 0; i < (int)ret.size(); i++)
120         ret[i] = (*this)[i] * r[i];
121     return ret;
122 }
123
124 FPS pre(int sz) const {
125     return FPS(begin(*this),
126                 begin(*this) + min((int)this->size(), sz));
127 }
128
129 FPS operator>>(int sz) const {
130     if ((int)this->size() <= sz) return {};
131     FPS ret(*this);
132     ret.erase(ret.begin(), ret.begin() + sz);
133     return ret;
134 }
135
136 FPS operator<<(int sz) const {
137     FPS ret(*this);
138     ret.insert(ret.begin(), sz, mint(0));
139     return ret;
140 }
141
142 FPS diff() const {
143     const int n = (int)this->size();
144     FPS ret(max(0, n - 1));
145     mint one(1), coeff(1);
146     for (int i = 1; i < n; i++) {
147         ret[i - 1] = (*this)[i] * coeff;
148         coeff += one;
149     }
150     return ret;
151 }

```

```

153 FPS integral() const {
154     const int n = (int)this->size();
155     FPS ret(n + 1);
156     ret[0] = mint(0);
157     if (n > 0) ret[1] = mint(1);
158     auto mod = mint::get_mod();
159     for (int i = 2; i <= n; i++)
160         ret[i] = (-ret[mod % i]) * (mod / i);
161     for (int i = 0; i < n; i++) ret[i + 1] *= (*this)[i];
162     return ret;
163 }
164
165 mint eval(mint x) const {
166     mint r = 0, w = 1;
167     for (auto &v : *this) r += w * v, w *= x;
168     return r;
169 }
170
171 FPS log(int deg = -1) const {
172     assert((*this)[0] == mint(1));
173     if (deg == -1) deg = (int)this->size();
174     return (this->diff() * this->inv(deg))
175         .pre(deg - 1)
176         .integral();
177 }
178
179 FPS pow(int64_t k, int deg = -1) const {
180     const int n = (int)this->size();
181     if (deg == -1) deg = n;
182     for (int i = 0; i < n; i++) {
183         if ((*this)[i] != mint(0)) {
184             if (i * k > deg) return FPS(deg, mint(0));
185             mint rev = mint(1) / (*this)[i];
186             FPS ret =
187                 (((*this) * rev) >> i).log(deg) * k).exp(deg) *
188                 ((*this)[i].pow(k));
189             ret = (ret << (i * k)).pre(deg);
190             if ((int)ret.size() < deg) ret.resize(deg, mint(0));
191             if ((*this)[i] != mint(0)) return ret;
192         }
193     }
194     return FPS(deg, mint(0));
195 }
196
197 static void *ntt_ptr;
198 static void set_fft();
199 FPS &operator=(const FPS &r);
200 void ntt();
201 void intt();
202 void ntt_doubling();
203 static int ntt_pr();
204 FPS inv(int deg = -1) const;
205 FPS exp(int deg = -1) const;
206 };
207 template <typename mint>
void *FormalPowerSeries<mint>::ntt_ptr = nullptr;

```

4.4. Theorems

4.4.1. Kirchhoff's Theorem

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

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

4.4.2. Tutte's Matrix

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

4.4.3. Cayley's Formula

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

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

spanning trees.

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

4.4.4. Erdős–Gallai Theorem

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

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

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

4.4.5. Burnside's Lemma

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

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

Then

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

5. Numeric

5.1. Barrett Reduction

```

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

```

5.2. Long Long Multiplication

```

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

```

5.3. Fast Fourier Transform

```

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

```

Requires: Mod Struct

```

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

```

5.4. Fast Walsh-Hadamard Transform

Requires: Mod Struct

```

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

```

5.5. Subset Convolution

Requires: Mod Struct

```

1 #pragma GCC target("popcnt")
2 #include <immintrin.h>
3
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(l << 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 }
35 }

```

5.6. Linear Recurrences

5.6.1. Berlekamp-Massey Algorithm

```

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

```

5.6.2. Linear Recurrence Calculation

```

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

```

5.7. Matrices

5.7.1. Determinant

Requires: Mod Struct

```

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

```

```

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    }
19    return ans;
}

```

5.7.2. Inverse

```

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

```

```

39     for (int i = n - 1; i > 0; --i) rep(j, 0, i) {
40         double v = A[j][i];
41         rep(k, 0, n) tmp[j][k] -= v * tmp[i][k];
42     }
43
44     rep(i, 0, n) rep(j, 0, n) A[col[i]][col[j]] = tmp[i][j];
45     return n;
46 }
47
48 int matInv_mod(vector<vector<ll>> &A) {
49     int n = sz(A);
50     vi col(n);
51     vector<vector<ll>> tmp(n, vector<ll>(n));
52     rep(i, 0, n) tmp[i][i] = 1, col[i] = i;
53
54     rep(i, 0, n) {
55         int r = i, c = i;
56         rep(j, i, n) rep(k, i, n) if (A[j][k]) {
57             r = j;
58             c = k;
59             goto found;
60         }
61     }
62     return i;
63 found:
64     A[i].swap(A[r]);
65     tmp[i].swap(tmp[r]);
66     rep(j, 0, n) swap(A[j][i], A[j][c]),
67     swap(tmp[j][i], tmp[j][c]);
68     swap(col[i], col[c]);
69     ll v = modpow(A[i][i], mod - 2);
70     rep(j, i + 1, n) {
71         ll f = A[j][i] * v % mod;
72         A[j][i] = 0;
73         rep(k, i + 1, n) A[j][k] =
74             (A[j][k] - f * A[i][k]) % mod;
75         rep(k, 0, n) tmp[j][k] =
76             (tmp[j][k] - f * tmp[i][k]) % mod;
77     }
78     rep(j, i + 1, n) A[i][j] = A[i][j] * v % mod;
79     rep(j, 0, n) tmp[i][j] = tmp[i][j] * v % mod;
80     A[i][i] = 1;
81
82     for (int i = n - 1; i > 0; --i) rep(j, 0, i) {
83         ll v = A[j][i];
84         rep(k, 0, n) tmp[j][k] =
85             (tmp[j][k] - v * tmp[i][k]) % mod;
86     }
87
88     rep(i, 0, n) rep(j, 0, n) A[col[i]][col[j]] =
89     tmp[i][j] % mod + (tmp[i][j] < 0 ? mod : 0);
90     return n;
91 }

```

5.7.3. Characteristic Polynomial

```

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

```

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

```

5.7.4. Solve Linear Equation

```

1

3 typedef vector<double> vd;
4 const double eps = 1e-12;
5
6 // solves for x: A * x = b
7 int solveLinear(vector<vd> &A, vd &b, vd &x) {
8     int n = sz(A), m = sz(x), rank = 0, br, bc;
9     if (n) assert(sz(A[0]) == m);
10    vi col(m);
11    iota(all(col), 0);
12
13    rep(i, 0, n) {
14        double v, bv = 0;
15        rep(r, i, n) rep(c, i, m) if ((v = fabs(A[r][c])) > bv)
16            br = r;
17        bc = c, bv = v;
18        if (bv <= eps) {
19            rep(j, i, n) if (fabs(b[j]) > eps) return -1;
20            break;
21        }
22        swap(A[i], A[br]);
23        swap(b[i], b[br]);
24        swap(col[i], col[bc]);
25        rep(j, 0, n) swap(A[j][i], A[j][bc]);
26        bv = 1 / A[i][i];
27        rep(j, i + 1, n) {
28            double fac = A[j][i] * bv;
29            b[j] -= fac * b[i];
30            rep(k, i + 1, m) A[j][k] -= fac * A[i][k];
31        }
32        rank++;
33    }
34
35    x.assign(m, 0);
36    for (int i = rank; i--;) {
37        b[i] /= A[i][i];
38        x[col[i]] = b[i];
39        rep(j, 0, i) b[j] -= A[j][i] * b[i];
40    }
41    return rank; // (multiple solutions if rank < m)
}

```

5.8. Polynomial Interpolation

```

1

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

```

5.9. 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 // // 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 + 1][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++)

```

```
101     if (B[i] == -1) {
102         int s = -1;
103         for (int j = 0; j <= n; j++)
104             if (s == -1 || D[i][j] < D[i][s] ||
105                 D[i][j] == D[i][s] && N[j] < N[s])
106                 s = j;
107         Pivot(i, s);
108     }
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 };
116
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 }
```

6. Geometry

6.1. Point

```

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

using pt = P<ll>;

6.1.1. Quaternion

```

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

6.1.2. Spherical Coordinates

```

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

6.2. Segments

```

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

6.3. Convex Hull

```

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

6.3.1. 3D Hull

```

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

```

31 rep(i, 0, 4) rep(j, i + 1, 4) rep(k, j + 1, 4)
32 mf(i, j, k, 6 - i - j - k);
33
34 rep(i, 4, sz(A)) {
35     rep(j, 0, sz(FS)) {
36         F f = FS[j];
37         if (f.q.dot(A[i]) > f.q.dot(A[f.a])) {
38             E(a, b).rem(f.c);
39             E(a, c).rem(f.b);
40             E(b, c).rem(f.a);
41             swap(FS[j-1], FS.back());
42             FS.pop_back();
43         }
44     }
45     int nw = sz(FS);
46     rep(j, 0, nw) {
47         F f = FS[j];
48         #define C(a, b, c)
49         if (E(a, b).cnt() != 2) mf(f.a, f.b, i, f.c);
50         C(a, b, c);
51         C(a, c, b);
52         C(b, c, a);
53     }
54     for (F &it : FS)
55         if ((A[it.b] - A[it.a])
56             .cross(A[it.c] - A[it.a])
57             .dot(it.q) <= 0)
58             swap(it.c, it.b);
59     return FS;
60 }
61

```

6.4. Angular Sort

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

6.5. Convex Polygon Minkowski Sum

```

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

```

6.6. Point In Polygon

```

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

```
13 } return cnt;
```

6.6.1. Convex Version

```

1 // no preprocessing version
2 // p must be a strict convex hull, counterclockwise
3 // if point is inside or on border
4 bool is_inside(const vector<pt> &c, pt p) {
5     int n = c.size(), l = 1, r = n - 1;
6     if (cross(c[0], c[1], p) < 0) return false;
7     if (cross(c[n - 1], c[0], p) < 0) return false;
8     while (l < r - 1) {
9         int m = (l + r) / 2;
10        T a = cross(c[0], c[m], p);
11        if (a > 0) l = m;
12        else if (a < 0) r = m;
13        else return dot(c[0] - p, c[m] - p) <= 0;
14    }
15    if (l == r) return dot(c[0] - p, c[l] - p) <= 0;
16    else return cross(c[l], c[r], p) >= 0;
17 }

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

6.6.2. Offline Multiple Points Version

Requires: Point, GNU PBDS

```
1
3
5
7
9 int n, m;
vector<Point> poly;
vector<Point> query;
vector<int> ans;
13
15 struct Segment {
16     Point a, b;
17     int id;
18 };
19 vector<Segment> segs;
20
21 Double Xnow;
22 inline Double get_y(const Segment &u, Double xnow = Xnow) {
23     const Point &a = u.a;
24     const Point &b = u.b;
25     return (a.y * (b.x - xnow) + b.y * (xnow - a.x)) /
26             (b.x - a.x);
27 }
28 bool operator<(Segment u, Segment v) {
29     Double yu = get_y(u);
30     Double yv = get_y(v);
31     if (yu != yv) return yu < yv;
32     return u.id < v.id;
33 }
34 ordered_map<Segment> st;
35
36 struct Event {
37     int type; // +1 insert seg, -1 remove seg, 0 query
38     Double x, y;
```

```

    int id;
39 };
40 bool operator<(Event a, Event b) {
41     if (a.x != b.x) return a.x < b.x;
42     if (a.type != b.type) return a.type < b.type;
43     return a.y < b.y;
44 }
45 vector<Event> events;
46
47 void solve() {
48     set<Double> xs;
49     set<Point> ps;
50     for (int i = 0; i < n; i++) {
51         xs.insert(poly[i].x);
52         ps.insert(poly[i]);
53     }
54     for (int i = 0; i < n; i++) {
55         Segment s{poly[i], poly[(i + 1) % n], i};
56         if (s.a.x > s.b.x ||
57             (s.a.x == s.b.x && s.a.y > s.b.y)) {
58             swap(s.a, s.b);
59         }
60         segs.push_back(s);
61
62         if (s.a.x != s.b.x) {
63             events.push_back({+1, s.a.x + 0.2, s.a.y, i});
64             events.push_back({-1, s.b.x - 0.2, s.b.y, i});
65         }
66     }
67     for (int i = 0; i < m; i++) {
68         events.push_back({0, query[i].x, query[i].y, i});
69     }
70     sort(events.begin(), events.end());
71     int cnt = 0;
72     for (Event e : events) {
73         int i = e.id;
74         Xnow = e.x;
75         if (e.type == 0) {
76             Double x = e.x;
77             Double y = e.y;
78             Segment tmp = {{x - 1, y}, {x + 1, y}, -1};
79             auto it = st.lower_bound(tmp);
80
81             if (ps.count(query[i]) > 0)
82                 ans[i] = 0;
83             else if (xs.count(x) > 0) {
84                 ans[i] = -2;
85             } else if (it != st.end() &&
86                         get_y(*it) == get_y(tmp)) {
87                 ans[i] = 0;
88             } else if (it != st.begin() &&
89                         get_y(*prev(it)) == get_y(tmp)) {
90                 ans[i] = 0;
91             } else {
92                 int rk = st.order_of_key(tmp);
93                 if (rk % 2 == 1) {
94                     ans[i] = 1;
95                 } else {
96                     ans[i] = -1;
97                 }
98             }
99         } else if (e.type == 1) {
100             st.insert(segs[i]);
101             assert((int)st.size() == ++cnt);
102         } else if (e.type == -1) {
103             st.erase(segs[i]);
104             assert((int)st.size() == --cnt);
105         }
106     }
107 }

```

6.7. Closest Pair

```

1 vector<pll> p; // sort by x first!
2 bool cmpy(const pll &a, const pll &b) const {
3     return a.y < b.y;
4 }
5 ll sq(ll x) { return x * x; }
6 // returns (minimum dist)^2 in [l, r)
7 ll solve(int l, int r) {
8     if (r - l <= 1) return le18;
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     }
21 }

```

6.8. Minimum Enclosing Circle

```

1
3     typedef Point<double> P;
4     double ccRadius(const P &A, const P &B, const P &C) {
5         return (B - A).dist() * (C - B).dist() * (A - C).dist() /
6             abs((B - A).cross(C - A)) / 2;
7     }
8     P ccCenter(const P &A, const P &B, const P &C) {
9         P b = C - A, c = B - A;
10        return A + (b * c.dist2() - c * b.dist2()).perp() /
11            b.cross(c) / 2;
12    }
13    pair<P, double> mec(vector<P> ps) {
14        shuffle(all(ps), mt19937(time(0)));
15        P o = ps[0];
16        double r = 0, EPS = 1 + 1e-8;
17        rep(i, 0, sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
18            o = ps[i], r = 0;
19            rep(j, 0, i) if ((o - ps[j]).dist() > r * EPS) {
20                o = (ps[i] + ps[j]) / 2;
21                r = (o - ps[i]).dist();
22                rep(k, 0, j) if ((o - ps[k]).dist() > r * EPS) {
23                    o = ccCenter(ps[i], ps[j], ps[k]);
24                    r = (o - ps[i]).dist();
25                }
26            }
27        }
28        return {o, r};
29    }

```

6.9. Delaunay Triangulation

```

1
3     typedef Point<ll> P;
4     typedef struct Quad *Q;
5     typedef __int128_t lll; // (can be ll if coords are < 2e4)
6     P arb(LLONG_MAX, LLONG_MAX); // not equal to any other point
7
8     struct Quad {
9         bool mark;
10        Q o, rot;
11        P p;
12        P F() { return r()>p; }
13        Q r() { return rot->rot; }
14        Q prev() { return rot->o->rot; }
15        Q next() { return r()>prev(); }
16    };
17
18    bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
19        lll p2 = p.dist2(), A = a.dist2() - p2,
20        B = b.dist2() - p2, C = c.dist2() - p2;
21        return p.cross(a, b) * C + p.cross(b, c) * A +
22               p.cross(c, a) * B >
23               0;
24    }
25    Q makeEdge(P orig, P dest) {
26        Q q[4] = {new Quad{0, 0, 0, orig}, new Quad{0, 0, 0, arb},
27                  new Quad{0, 0, 0, dest}, new Quad{0, 0, 0, arb}};
28        rep(i, 0, 4) q[i]->o = q[-i & 3],
29                      q[i]->rot = q[(i + 1) & 3];
30        return *q;
31    }
32    void splice(Q a, Q b) {
33        swap(a->o->rot->o, b->o->rot->o);
34        swap(a->o, b->o);
35    }
36    Q connect(Q a, Q b) {
37        Q q = makeEdge(a->F(), b->p);
38        splice(q, a->next());
39        splice(q->r(), b);
40        return q;
41    }
42
43    pair<Q, Q> rec(const vector<P> &s) {
44        if (sz(s) <= 3) {
45            Q a = makeEdge(s[0], s[1]),
46            b = makeEdge(s[1], s.back());
47            if (sz(s) == 2) return {a, a->r()};
48            splice(a->r(), b);
49            auto side = s[0].cross(s[1], s[2]);
50            Q c = side ? connect(b, a) : 0;
51            return {side < 0 ? c->r() : a, side < 0 ? c : b->r()};
52        }
53    }

```

```

55 #define H(e) e->F(), e->p
56 #define valid(e) (e->F()).cross(H(base)) > 0
57 Q A, B, ra, rb;
58 int half = sz(s) / 2;
59 tie(ra, A) = rec({all(s) - half});
60 tie(B, rb) = rec({sz(s) - half + all(s)});
61 while ((B->p.cross(H(A)) < 0 && (A = A->next()) || 
62         (A->p.cross(H(B)) > 0 && (B = B->r()-o))) ||
63         ;
64 Q base = connect(B->r(), A);
65 if (A->p == ra->p) ra = base->r();
66 if (B->p == rb->p) rb = base;
67 #define DEL(e, init, dir)
68 Q e = init->dir;
69 if (valid(e))
70     while (circ(e->dir->F(), H(base), e->F())) {
71         Q t = e->dir;
72         splice(e, e->prev());
73         splice(e->r(), e->r()->prev());
74         e = t;
75     }
76 for (;;) {
77     DEL(LC, base->r(), o);
78     DEL(RC, base, prev());
79     if (!valid(LC) && !valid(RC)) break;
80     if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
81         base = connect(RC, base->r());
82     else base = connect(base->r(), LC->r());
83 }
84 return {ra, rb};
85 }

86 // returns [A_0, B_0, C_0, A_1, B_1, ...]
87 // where A_i, B_i, C_i are counter-clockwise triangles
88 vector<P> triangulate(vector<P> pts) {
89     sort(all(pts));
90     assert(unique(all(pts)) == pts.end());
91     if (sz(pts) < 2) return {};
92     Q e = rec(pts).first;
93     vector<Q> q = {e};
94     int qi = 0;
95     while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
96 #define ADD
97 {
98     Q c = e;
99     do {
100         c->mark = 1;
101         pts.push_back(c->p);
102         q.push_back(c->r());
103         c = c->next();
104     } while (c != e);
105 }
106 ADD;
107 pts.clear();
108 while (qi < sz(q))
109     if (!(e = q[qi++])->mark) ADD;
110 return pts;
111 }
```

```

11     }
12     Point GetIntersection(Line a, Line b) {
13         Vector u = a.P - b.P;
14         Double t = Cross(b.v, u) / Cross(a.v, b.v);
15         return a.P + a.v * t;
16     }
17     int HalfplaneIntersection(Line *L, int n, Point *poly) {
18         sort(L, L + n);
19         int first, last;
20         Point *p = new Point[n];
21         Line *q = new Line[n];
22         q[first = last = 0] = L[0];
23         for (int i = 1; i < n; i++) {
24             while (first < last && !OnLeft(L[i], p[last - 1]))
25                 last--;
26             while (first < last && !OnLeft(L[i], p[first])) first++;
27             q[++last] = L[i];
28             if (fabs(Cross(q[last].v, q[last - 1].v)) < EPS) {
29                 last--;
30                 if (OnLeft(q[last], L[i].P)) q[last] = L[i];
31             }
32             if (first < last)
33                 p[last - 1] = GetIntersection(q[last - 1], q[last]);
34         }
35         while (first < last && !OnLeft(q[first], p[last - 1]))
36             last--;
37         if (last - first <= 1) return 0;
38         p[last] = GetIntersection(q[last], q[first]);
39         int m = 0;
40         for (int i = first; i <= last; i++) poly[m++] = p[i];
41     }
42 }
```

6.9.1. Slower Version

```

1

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

6.10. Half Plane Intersection

```

1 struct Line {
2     Point P;
3     Vector v;
4     bool operator<(const Line &b) const {
5         return atan2(v.y, v.x) < atan2(b.v.y, b.v.x);
6     }
7 };
8 bool OnLeft(const Line &L, const Point &p) {
9     return Cross(L.v, p - L.P) > 0;
```

7. Strings

7.1. Knuth-Morris-Pratt Algorithm

```

1
3 vector<int> pi(const string &s) {
4     vector<int> p(s.size());
5     for (int i = 1; i < s.size(); i++) {
6         int g = p[i - 1];
7         while (g && s[i] != s[g]) g = p[g - 1];
8         p[i] = g + (s[i] == s[g]);
9     }
10    return p;
11}
12
13 vector<int> match(const string &s, const string &pat) {
14    vector<int> p = pi(pat + '\0' + s), res;
15    for (int i = p.size() - s.size(); i < p.size(); i++)
16        if (p[i] == pat.size())
17            res.push_back(i - 2 * pat.size());
18    return res;
19}

```

7.2. Aho-Corasick Automaton

```

1 struct Aho_Corasick {
2     static const int maxc = 26, maxn = 4e5;
3     struct NODES {
4         int Next[maxc], fail, ans;
5     };
6     NODES T[maxn];
7     int top, qtop, q[maxn];
8     int get_node(const int &fail) {
9         fill_n(T[top].Next, maxc, 0);
10        T[top].fail = fail;
11        T[top].ans = 0;
12        return top++;
13    }
14    int insert(const string &s) {
15        int ptr = 1;
16        for (char c : s) { // change char id
17            c -= 'a';
18            if (!T[ptr].Next[c]) T[ptr].Next[c] = get_node(ptr);
19            ptr = T[ptr].Next[c];
20        }
21        return ptr;
22    } // return ans_last_place
23    void build_fail(int ptr) {
24        int tmp;
25        for (int i = 0; i < maxc; i++) {
26            if (T[ptr].Next[i]) {
27                tmp = T[ptr].fail;
28                while (tmp != 1 && !T[tmp].Next[i])
29                    tmp = T[tmp].fail;
30                if (T[tmp].Next[i] != T[ptr].Next[i])
31                    if (T[ptr].Next[i]) tmp = T[ptr].Next[i];
32                T[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     }
68 }

```

```

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 }

```

7.3. Suffix Array

```

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

```

7.4. Suffix Tree

```

1 struct SAM {
2     static const int maxc = 26; // char range
3     static const int maxn = 10010; // string len
4     struct Node {
5         Node *green, *edge[maxc];
6         int max_len, in, times;
7     } *root, *last, reg[maxn * 2];
8     int top;
9     Node *get_node(int _max) {
10        Node *re = &reg[top++];
11        re->in = 0, re->times = 1;
12        re->max_len = _max, re->green = 0;
13        for (int i = 0; i < maxc; i++) re->edge[i] = 0;
14        return re;
15    }
16    void insert(const char c) { // c in range [0, maxc)
17        Node *p = last;
18        last = get_node(p->max_len + 1);
19        while (p && !p->edge[c])
20            p->edge[c] = last, p = p->green;
21        if (!p) last->green = root;
22        else {
23            Node *pot_green = p->edge[c];
24            if ((pot_green->max_len) == (p->max_len + 1))
25                last->green = pot_green;
26            else {
27                Node *wish = get_node(p->max_len + 1);
28                wish->times = 0;
29                while (p && p->edge[c] == pot_green)
30                    p->edge[c] = wish, p = p->green;
31                for (int i = 0; i < maxc; i++)
32                    wish->edge[i] = pot_green->edge[i];
33                wish->green = pot_green->green;
34                pot_green->green = wish;
35                last->green = wish;
36            }
37        }
38    }
39 }

```

```

39 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 void build(const string &s) {
53     top = 0;
54     root = last = get_node(0);
55     for (char c : s) insert(c - 'a'); // change char id
56     get_times(root);
57 }
58 // call build before solve
59 int solve(const string &s) {
60     Node *p = root;
61     for (char c : s)
62         if (!(p = p->edge[c - 'a'])) // change char id
63             return 0;
64     return p->times;
65 }

```

7.5. Cocke-Younger-Kasami Algorithm

```

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

```

```

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

```

7.6. Z Value

```

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

```

7.7. Manacher's Algorithm

```

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

```

7.8. Minimum Rotation

```

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

```

```

11     a = b;
12     break;
13   }
14 }
15 return a;
16 }
```

7.9. Palindromic Tree

```

1
3 struct palindromic_tree {
4   struct node {
5     int next[26], fail, len;
6     int cnt,
7     num; // cnt: appear times, num: number of pal. suf.
8     node(int l = 0) : fail(0), len(l), cnt(0), num(0) {
9       for (int i = 0; i < 26; ++i) next[i] = 0;
10    }
11  };
12  vector<node> St;
13  vector<char> s;
14  int last, n;
15  palindromic_tree() : St(2), last(1), n(0) {
16    St[0].fail = 1, St[1].len = -1, s.pb(-1);
17  }
18  inline void clear() {
19    St.clear(), s.clear(), last = 1, n = 0;
20    St.pb(0), St.pb(-1);
21    St[0].fail = 1, s.pb(-1);
22  }
23  inline int get_fail(int x) {
24    while (s[n - St[x].len - 1] != s[n]) x = St[x].fail;
25    return x;
26  }
27  inline void add(int c) {
28    s.push_back(c -= 'a'), ++n;
29    int cur = get_fail(last);
30    if (!St[cur].next[c]) {
31      int now = SZ(St);
32      St.pb(St[cur].len + 2);
33      St[now].fail = St[get_fail(St[cur].fail)].next[c];
34      St[cur].next[c] = now;
35      St[now].num = St[St[now].fail].num + 1;
36    }
37    last = St[cur].next[c], ++St[last].cnt;
38  }
39  inline void count() { // counting cnt
40    auto i = St.rbegin();
41    for (; i != St.rend(); ++i) {
42      St[i->fail].cnt += i->cnt;
43    }
44  }
45  inline int size() { // The number of diff. pal.
46    return SZ(St) - 2;
47  }
48};
```

8. Debug List

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