

MeloN's XCPC Library - ICPC2024 - Kunming



Library

目录

- Library
 - 目录
 - Z_H
 - MeloN_H.hpp
 - MeloN_IO.hpp
 - MeloN_PRET.hpp
 - MeloN_debug.hpp
 - fast_io.hpp
 - ds
 - LinearBasis.hpp
 - Wavelet_Matrix.hpp
 - bit_vec.hpp
 - chothlly.hpp
 - dsu.hpp
 - fenw.hpp
 - hashmap.hpp
 - heap.hpp
 - rollback_array.hpp
 - rollback_dsu.hpp
 - splay.hpp
 - sqrt_tree.hpp
 - st_table.hpp
 - rollback mo
 - block
 - flow
 - max_flow.hpp
 - max_flow_min_cost.hpp
 - geo
 - 两圆面积覆盖
 - 正n角形面积
 - 正n锥体体积
 - 1-base.hpp
 - 2-apollonian_circle.hpp
 - 3-angle_sort.hpp
 - 4-closest_pair.hpp
 - 5-hull.hpp
 - 6-convex_polygon.hpp
 - 7-points_in_triangles.hpp
 - 8-distance.hpp
 - 9-furthest_pair.hpp
 - 10-triangle_area.hpp
 - 11-in_circle.hpp

- 12-line_inside_polygon.hpp
- 13-manhattan_mst.hpp
- 14-max_norm_sum.hpp
- 15-minkowski_sum.hpp
- 16-out_circle.hpp
- 17-minimum_enclosing_circle.hpp
- graph
 - 2_sat.hpp
 - dijkstra.hpp
 - 三元环计数
 - 最大团
- math
 - exgcd.hpp
 - mat.hpp
 - prims_set.hpp
 - radix_sort.hpp
 - sieve.hpp
- random
 - random.hpp
- string
 - SA.hpp
 - SAM.hpp
 - acam.hpp
 - hash.hpp
 - manache.hpp
 - trie
- tree
 - LCA.hpp
 - LTT.hpp
 - centroid.hpp
 - unrooted_tree_hash.hpp
 - 最小斯坦纳树
- a_monoid
 - max_add.hpp
 - min_add.hpp
 - minidx_add.hpp
 - sum_add.hpp
- monoid
 - add.hpp
 - add_array.hpp
 - add_pair.hpp
 - gcd.hpp
 - max.hpp
 - max_idx.hpp
 - min.hpp

- min_idx.hpp
- sum.hpp
- xor.hpp
- seg
 - lazy_seg_base.hpp
 - seg_base.hpp
- mod
 - modint.hpp
- others
 - 快速取模
 - date time

Z_H

MeloN_H.hpp

```
1 using std::array, std::bitset, std::deque, std::greater, std::less, std::map,
2     std::multiset, std::pair, std::priority_queue, std::set, std::stack,
3     std::string, std::vector;
4
5 using NAME = void;      using uint = unsigned;   using ll = long long;      using ull = unsigned long long;
6 using ld = long double; using i128 = __int128_t; using u128 = __uint128_t; using f128 = __float128;
7
8 #define meion      auto
9 #define iroha      return
```

MeloN_IO.hpp

```
1 namespace MeIoN_IO {
2     std::istream& operator>>(std::istream& is, i128& n) {
3         string s;
4         is >> s;
5         int f = s[0] == '-';
6         n = 0;
7         for (int i = f; i < s.length(); ++i) {
8             n = n * 10 + s[i] - '0';
9         }
10        if (f) n = -n;
11        return is;
12    }
13    std::ostream& operator<<(std::ostream& os, i128 n) {
14        string s;
15        bool f = n < 0;
16        if (f) n = -n;
17        while (n) s += '0' + n % 10, n /= 10;
18        if (s.empty()) s += '0';
19        if (f) s += '-';
20        std::reverse(s.begin(), s.end());
21        return os << s;
22    }
23    std::istream& operator>>(std::istream& is, f128& n) {
24        string s;
25        is >> s;
26        n = std::stold(s);
27        return is;
28    }
```

```
29 } using namespace MeIoN_IO;
```

MeloN_PRET.hpp

```
1 namespace MeIoN_Pre_Things {
2     int T = 1;
3     std::mt19937 rng(std::chrono::steady_clock::now().time_since_epoch().count());
4     std::mt19937_64 rng_64(std::chrono::steady_clock::now().time_since_epoch().count());
5     constexpr int mod99 = 998244353, mod17 = 1000000007;
6     constexpr ld eps = 1E-8L, pi = 3.1415926535897932384626433832795L;
7     template <class T>
8     constexpr T inf = 0;
9     template <>
10    constexpr int inf<int> = 2147483647;
11    template <>
12    constexpr uint inf<uint> = 4294967294U;
13    template <>
14    constexpr ll inf<ll> = 9223372036854775807LL;
15    template <>
16    constexpr ull inf<ull> = 18446744073709551614ULL;
17    template <>
18    constexpr i128 inf<i128> = i128(inf<ll>) * 2'000'000'000'000'000'000;
19    template <>
20    constexpr double inf<double> = inf<ll>;
21    template <>
22    constexpr long double inf<long double> = inf<ll>;
23    template <typename T>
24    inline T lowbit(T x) { return x & -x; }
25    template <typename T>
26    inline int popcount(T n) { return std::__popcount(n); }
27    template <typename T>
28    inline int clz(T n) { return std::__countl_zero(n); }
29    template <class T, class S>
30    inline bool chmax(T &a, const S &b) {
31        return (a < b ? a = b, 1 : 0);
32    }
33    template <class T, class S>
34    inline bool chmin(T &a, const S &b) {
35        return (a > b ? a = b, 1 : 0);
36    }
37    template <typename T>
38    std::vector<int> argsort(const std::vector<T> &A) {
39        std::vector<int> ids(A.size());
40        std::iota(ids.begin(), ids.end(), 0);
41        std::sort(ids.begin(), ids.end(), [&](int i, int j) { return A[i] < A[j] or (A[i] == A[j] and i < j); });
42        return ids;
43    }
44    template <typename T>
45    vector<T> rearrange(const vector<T> &A, const vector<int> &I) {
46        vector<T> B(I.size());
47        for (int i = 0, ed = I.size(); i < ed; ++i)
48            B[i] = A[I[i]];
49        return B;
50    }
51    // (0, 1, 2, 3, 4) -> (-1, 0, 1, 1, 2)
52    int topbit(int x) { return (x == 0 ? -1 : 31 - __builtin_clz(x)); }
53    int topbit(uint x) { return (x == 0 ? -1 : 31 - __builtin_clz(x)); }
54    int topbit(ll x) { return (x == 0 ? -1 : 63 - __builtin_clzll(x)); }
55    int topbit(ull x) { return (x == 0 ? -1 : 63 - __builtin_clzll(x)); }
56    template <typename T, typename U>
57    inline T ceil(T x, U y) { return (x > 0 ? (x + y - 1) / y : x / y); }
58    template <typename T, typename U>
```

```

59 inline T floor(T x, U y) { return (x > 0 ? x / y : (x - y + 1) / y); }
60 template <typename F>
61 ll binary_search(F check, ll ok, ll ng, bool check_ok = true) {
62     if (check_ok) assert(check(ok));
63     while (std::abs(ok - ng) > 1) {
64         auto x = (ng + ok) / 2;
65         (check(x) ? ok : ng) = x;
66     }
67     return ok;
68 }
69 template <class T>
70 struct MeIoN_Queue {
71     vector<T> q;
72     int pos = 0;
73     void reserve(int n) { q.reserve(n); }
74     int size() const { return int(q.size()) - pos; }
75     bool empty() const { return pos == int(q.size()); }
76     T& front() { return q[pos]; }
77     T& back() { return q.back(); }
78     template <typename... Args>
79     void emplace_back(Args&&... args) {
80         q.emplace_back(std::forward<Args>(args)...);
81     }
82     void push_back(const T& v) { q.push_back(v); }
83     void pop() { ++pos; }
84     void pop_back() { q.pop_back(); }
85     void clear() {
86         q.clear();
87         pos = 0;
88     }
89 };
90 } using namespace MeIoN_Pre_Things;

```

MeloN_debug.hpp

```

1 // copy from https://github.com/Heltion/debug.h
2 template <class T, size_t size = std::tuple_size<T>::value>
3 std::string to_debug(T, std::string s = "")
4     requires(not std::ranges::range<T>);
5 std::string to_debug(auto x)
6     requires requires(std::ostream& os) { os << x; }
7 {
8     return static_cast<std::ostringstream>(std::ostringstream() << x).str();
9 }
10 std::string to_debug(std::ranges::range auto x, std::string s = "")
11     requires(not std::is_same_v<decltype(x), std::string>)
12 {
13     for (auto xi : x) {
14         s += ", " + to_debug(xi);
15     }
16     return "[" + s.substr(s.empty() ? 0 : 2) + "]";
17 }
18 template <class T, size_t size>
19 std::string to_debug(T x, std::string s)
20     requires(not std::ranges::range<T>)
21 {
22     [&<size_t... I>(std::index_sequence<I...>) {
23         ((s += ", " + to_debug(std::get<I>(x))), ...);
24     }(std::make_index_sequence<size>());
25     return "(" + s.substr(s.empty() ? 0 : 2) + ")";
26 }

```



```
27 #ifdef MeIoN
28 #define debug(...) std::cout << "Ciallo~(∠ · ω< )∩★ " \
29 << "(" #__VA_ARGS__ ") = " \
30 << to_debug(std::tuple(__VA_ARGS__)) \
31 << std::endl;
32 #else
33 #define debug(...) void(0721)
34 #endif
```

fast_io.hpp

```
1 namespace fast_io {
2     static constexpr uint32_t SZ = 1 << 17;
3     char ibuf[SZ];
4     char obuf[SZ];
5     char out[100];
6     // pointer of ibuf, obuf
7     uint32_t pil = 0, pir = 0, por = 0;
8
9     struct Pre {
10         char num[10000][4];
11         constexpr Pre() : num() {
12             for (int i = 0; i < 10000; i++) {
13                 int n = i;
14                 for (int j = 3; j >= 0; j--) {
15                     num[i][j] = n % 10 | '0';
16                     n /= 10;
17                 }
18             }
19         }
20     } constexpr pre;
21
22     inline void load() {
23         memcpy(ibuf, ibuf + pil, pir - pil);
24         pir = pir - pil + fread(ibuf + pir - pil, 1, SZ - pir + pil, stdin);
25         pil = 0;
26         if (pir < SZ) ibuf[pir++] = '\n';
27     }
28
29     inline void flush() {
30         fwrite(obuf, 1, por, stdout);
31         por = 0;
32     }
33     void rd(char &c) {
34         do {
35             if (pil + 1 > pir) load();
36             c = ibuf[pil++];
37         } while (isspace(c));
38     }
39
40     void rd(string &x) {
41         x.clear();
42         char c;
43         do {
44             if (pil + 1 > pir) load();
45             c = ibuf[pil++];
46         } while (isspace(c));
47         do {
48             x += c;
49             if (pil == pir) load();
50             c = ibuf[pil++];
51         } while (!isspace(c));
```

```

52     }
53
54     template <typename T>
55     void rd_real(T &x) {
56         string s;
57         rd(s);
58         x = stod(s);
59     }
60
61     template <typename T>
62     void rd_integer(T &x) {
63         if (pil + 100 > pir) load();
64         char c;
65         do c = ibuf[pil++];
66         while (c < '-');
67         bool minus = 0;
68         if constexpr (std::is_signed<T>::value || std::is_same_v<T, i128>) {
69             if (c == '-') {
70                 minus = 1, c = ibuf[pil++];
71             }
72         }
73         x = 0;
74         while ('0' <= c) {
75             x = x * 10 + (c & 15), c = ibuf[pil++];
76         }
77         if constexpr (std::is_signed<T>::value || std::is_same_v<T, i128>) {
78             if (minus) x = -x;
79         }
80     }
81
82     void rd(int &x) { rd_integer(x); }
83     void rd(ll &x) { rd_integer(x); }
84     void rd(i128 &x) { rd_integer(x); }
85     void rd(uint &x) { rd_integer(x); }
86     void rd(ull &x) { rd_integer(x); }
87     void rd(u128 &x) { rd_integer(x); }
88     void rd(double &x) { rd_real(x); }
89     void rd(long double &x) { rd_real(x); }
90     void rd(f128 &x) { rd_real(x); }
91
92     void read() {}
93     template <class H, class... T>
94     void read(H &h, T &...t) {
95         rd(h), read(t...);
96     }
97
98     void wt(const char c) {
99         if (por == SZ) flush();
100        obuf[por++] = c;
101    }
102    void wt(const string s) {
103        for (char c : s) wt(c);
104    }
105    void wt(const char *s) {
106        size_t len = strlen(s);
107        for (size_t i = 0; i < len; i++) wt(s[i]);
108    }
109
110    template <typename T>
111    void wt_integer(T x) {
112        if (por > SZ - 100) flush();
113        if (x < 0) {

```



```

114         obuf[por++] = '-', x = -x;
115     }
116     int outi;
117     for (outi = 96; x >= 10000; outi -= 4) {
118         memcpy(out + outi, pre.num[x % 10000], 4);
119         x /= 10000;
120     }
121     if (x >= 1000) {
122         memcpy(obuf + por, pre.num[x], 4);
123         por += 4;
124     } else if (x >= 100) {
125         memcpy(obuf + por, pre.num[x] + 1, 3);
126         por += 3;
127     } else if (x >= 10) {
128         int q = (x * 103) >> 10;
129         obuf[por] = q | '0';
130         obuf[por + 1] = (x - q * 10) | '0';
131         por += 2;
132     } else
133         obuf[por++] = x | '0';
134     memcpy(obuf + por, out + outi + 4, 96 - outi);
135     por += 96 - outi;
136 }
137
138 template <typename T>
139 void wt_real(T x) {
140     std::ostringstream oss;
141     oss << std::fixed << std::setprecision(15) << double(x);
142     std::string s = oss.str();
143     wt(s);
144 }
145
146 void wt(int x) { wt_integer(x); }
147 void wt(ll x) { wt_integer(x); }
148 void wt(i128 x) { wt_integer(x); }
149 void wt(uint x) { wt_integer(x); }
150 void wt(ull x) { wt_integer(x); }
151 void wt(u128 x) { wt_integer(x); }
152 void wt(double x) { wt_real(x); }
153 void wt(long double x) { wt_real(x); }
154 void wt(f128 x) { wt_real(x); }
155 void wt(std::ios_base &(__pf)(std::ios_base &)) {}
156 void wt(const std::_Setprecision &x) {}
157
158 // gcc expansion. called automaticall after main.
159 void __attribute__((destructor)) _d() { flush(); }
160
161 struct io_auxiliary {
162     void sync_with_stdio(bool ok) { }
163 } *auxiliary_io;
164
165 struct meion_fast_io {
166     template<typename T>
167     friend meion_fast_io& operator>>(meion_fast_io &in, T &c) {
168         rd(c);
169         return in;
170     }
171     template<typename T>
172     friend meion_fast_io& operator<<(meion_fast_io &out, const T &c) {
173         wt(c);
174         return out;
175     }

```

```
176         io_auxiliary *tie(std::nullptr_t x) {
177             return auxiliary_io;
178         }
179     } fin, fout;
180 }
181 #define fast
182 #define cin fin
183 #define cout fout
184 namespace std {
185     using fast_io::cin, fast_io::cout;
186 }
```

ds

LinearBasis.hpp

```
1  struct LinearBasis {
2      static const int B = 30;
3      LinearBasis() { memset(basis, -1, sizeof(basis)); }
4      void add(int v) {
5          v = ask(v);
6          if (v) {
7              int pivot = 30 - clz(v);
8              for (int i = 0; i < B; ++i) {
9                  if (~basis[i] && (basis[i] >> pivot & 1)) {
10                     basis[i] ^= v;
11                 }
12             }
13             basis[pivot] = v;
14         }
15     }
16     int ask(int v) const {
17         for (int i = B; i--;) {
18             if ((v >> i & 1) and ~basis[i]) {
19                 v ^= basis[i];
20             }
21         }
22         return v;
23     }
24     int basis[B];
25 };
26 struct LinearBasis_64 {
27     static const int B = 63;
28     LinearBasis_64() { memset(basis, -1, sizeof(basis)); }
29     void add(ll v) {
30         v = ask(v);
31         if (v) {
32             int pivot = 62 - clz(v);
33             for (int i = 0; i < B; ++i) {
34                 if (~basis[i] && (basis[i] >> pivot & 1)) {
35                     basis[i] ^= v;
36                 }
37             }
38             basis[pivot] = v;
39         }
40     }
41     ll ask(ll v) const {
42         for (int i = B; i--;) {
43             if ((v >> i & 1) and ~basis[i]) {
44                 v ^= basis[i];
45             }
46         }
47         return v;
48     }
49     ll quis(ll v = 0ll) {
50         for (int i = B; i--;) {
51             if (not (v >> i & 1) and ~basis[i]) {
52                 v ^= basis[i];
53             }
54         }
55         return v;
56     }
57     ll basis[B];
```

```
58 };
```

Wavelet_Matrix.hpp

```
1  struct Bit_Vector {
2      vector<pair<unsigned, unsigned>> dat;
3      Bit_Vector(int n) { dat.assign((n + 63) >> 5, {0, 0}); }
4      void set(int i) { dat[i >> 5].first |= unsigned(1) << (i & 31); }
5      void build() {
6          for (int i = 0, ed = int(dat.size()) - 1; i < ed; ++i)
7              dat[i + 1].second = dat[i].second + std::popcount(dat[i].first);
8      }
9      // [0, k) 内の 1 の個数
10     int rank(int k, bool f = 1) {
11         auto[a, b] = dat[k >> 5];
12         int ret = b + std::popcount(a & ((unsigned(1) << (k & 31)) - 1));
13         return (f ? ret : k - ret);
14     }
15 };
16 // 座圧するかどうかを COMPRESS で指定する
17 // xor 的な使い方をする場合には、コンストラクタで Log を渡すこと
18 template <typename T = int, bool COMPRESS = false>
19 struct Wavelet_Matrix {
20     int N, lg;
21     vector<int> mid;
22     vector<Bit_Vector> bv;
23     vector<T> key;
24     bool set_log;
25     Wavelet_Matrix(vector<T> A, int log = -1)
26         : N(A.size()), lg(log), set_log(log != -1) {
27         if (COMPRESS) {
28             assert(!set_log);
29             key.reserve(N);
30             vector<int> I = argsort(A);
31             for (auto&& i : I) {
32                 if (key.empty() || key.back() != A[i]) key.emplace_back(A[i]);
33                 A[i] = (int)key.size() - 1;
34             }
35             key.shrink_to_fit();
36         }
37         if (lg == -1) lg = std::__lg(std::max<ll>(qmax(A), 1)) + 1;
38         mid.resize(lg);
39         bv.assign(lg, Bit_Vector(N));
40         vector<T> A0(N), A1(N);
41         for (ll d = (lg)-1; d >= ll(0); --d) {
42             int p0 = 0, p1 = 0;
43             for (ll i = 0; i < ll(N); ++i) {
44                 bool f = (A[i] >> d & 1);
45                 if (!f) A0[p0++] = A[i];
46                 if (f) bv[d].set(i), A1[p1++] = A[i];
47             }
48             mid[d] = p0;
49             bv[d].build();
50             std::swap(A, A0);
51             for (ll i = 0; i < ll(p1); ++i) A[p0 + i] = A1[i];
52         }
53     }
54     // xor した結果で [a, b) に収まるものを数える
55     int count(int L, int R, T a, T b, T xor_val = 0) {
56         return prefix_count(L, R, b, xor_val) - prefix_count(L, R, a, xor_val);
57     }
58     // xor した結果で [0, x) に収まるものを数える
```

```

59     int prefix_count(int L, int R, T x, T xor_val = 0) {
60         if (xor_val != 0) assert(set_log);
61         x = (COMPRESS
62             ? std::distance((key).begin(),
63                           std::lower_bound(key.begin(), key.end(), (x)))
64             : x);
65         if (x >= (1 << lg)) return R - L;
66         int ret = 0;
67         for (int d = lg - 1; d >= 0; --d) {
68             bool add = (x >> d) & 1;
69             bool f = ((x ^ xor_val) >> d) & 1;
70             if (add) ret += bv[d].rank(R, !f) - bv[d].rank(L, !f);
71             L = bv[d].rank(L, f) + (f ? mid[d] : 0);
72             R = bv[d].rank(R, f) + (f ? mid[d] : 0);
73         }
74         return ret;
75     }
76     T kth(int L, int R, int k, T xor_val = 0) { // k : 0 index
77         if (xor_val != 0) assert(set_log);
78         assert(0 <= k && k < R - L);
79         T ret = 0;
80         for (int d = lg - 1; d >= 0; --d) {
81             bool f = (xor_val >> d) & 1;
82             int l0 = bv[d].rank(L, 0), r0 = bv[d].rank(R, 0);
83             int kf = (f ? (R - L) - (r0 - l0) : (r0 - l0));
84             if (k < kf) {
85                 if (!f) L = l0, R = r0;
86                 if (f) L += mid[d] - l0, R += mid[d] - r0;
87             } else {
88                 k -= kf, ret |= T(1) << d;
89                 if (!f) L += mid[d] - l0, R += mid[d] - r0;
90                 if (f) L = l0, R = r0;
91             }
92         }
93         return (COMPRESS ? key[ret] : ret);
94     }
95 };

```

bit_vec.hpp

```

1  template <const int N>
2  struct bitarray {
3      static constexpr int sz = ((N + 127) >> 6);
4      array<ull, sz> v;
5      void set(int i) {
6          v[i >> 6] |= 1ull << (i & 63);
7      }
8      void reset(int i) {
9          v[i >> 6] &= ~(1ull << (i & 63));
10     }
11     void reset() {
12         std::ranges::fill(v, 0ull);
13     }
14     bool operator[](int i) {
15         return v[i >> 6] >> (i & 63) & 1;
16     }
17     bitarray operator &=(const bitarray &b) {
18         for (int i = 0, ed = sz; i < ed; ++i) {
19             v[i] &= b.v[i];
20         }
21         return *this;
22     }

```

```
23     bitarray operator |=(const bitarray &b) {
24         for (int i = 0, ed = sz; i < ed; ++i) {
25             v[i] |= b.v[i];
26         }
27         return *this;
28     }
29     bitarray operator ^=(const bitarray &b) {
30         for (int i = 0, ed = sz; i < ed; ++i) {
31             v[i] ^= b.v[i];
32         }
33         return *this;
34     }
35     bitarray operator &(const bitarray &b) {
36         return bitarray(*this) &= b;
37     }
38     bitarray operator |(const bitarray &b) {
39         return bitarray(*this) |= b;
40     }
41     bitarray operator ^(const bitarray &b) {
42         return bitarray(*this) ^= b;
43     }
44     bitarray operator ~() const {
45         bitarray ret(*this);
46         for (int i = 0, ed = sz; i < ed; ++i) {
47             ret.v[i] = ~ret.v[i];
48         }
49         return ret;
50     }
51     bitarray operator <<=(const int t) {
52         bitarray ret;
53         ret.v.fill(0ull);
54         ull last = 0;
55         int high = t >> 6, low = t & 63;
56         for(int i = 0; i + high < sz; ++i) {
57             ret.v[i + high] = last | (v[i] << low);
58             if (low) last = v[i] >> (64 - low);
59         }
60         return (*this) = ret;
61     }
62     bitarray operator >>=(const int t) {
63         bitarray ret;
64         ret.v.fill(0ull);
65         ull last = 0;
66         int high = t >> 6, low = t & 63;
67         for(int i = int(v.size() - 1); i > high - 1; --i) {
68             ret.v[i - high] = last | (v[i] >> low);
69             if (low) last = v[i] << (64 - low);
70         }
71         return (*this) = ret;
72     }
73     bitarray operator <<(const int t) {
74         return bitarray(*this) <<= t;
75     }
76     bitarray operator >>(const int t) {
77         return bitarray(*this) >>= t;
78     }
79     std::string to_string() {
80         std::string ans;
81         for (int i = 0; i < N; ++i) {
82             ans += '0' + (*this)[i];
83         }
84         return ans;
```

```
85     }
86 };
87 struct bitvector {
88     int n;
89     vector<ull> v;
90     bitvector(int n) : n(n), v(n + 127 >> 6, 0ull) {}
91     void set(int i) {
92         v[i >> 6] |= 1ull << (i & 63);
93     }
94     void reset(int i) {
95         v[i >> 6] &= ~(1ull << (i & 63));
96     }
97     void reset() {
98         std::ranges::fill(v, 0ull);
99     }
100    bool operator[](int i) {
101        return v[i >> 6] >> (i & 63) & 1;
102    }
103    bitvector operator &=(const bitvector &b) {
104        for (int i = 0, ed = int(v.size()); i < ed; ++i) {
105            v[i] &= b.v[i];
106        }
107        return *this;
108    }
109    bitvector operator |=(const bitvector &b) {
110        for (int i = 0, ed = int(v.size()); i < ed; ++i) {
111            v[i] |= b.v[i];
112        }
113        return *this;
114    }
115    bitvector operator ^=(const bitvector &b) {
116        for (int i = 0, ed = int(v.size()); i < ed; ++i) {
117            v[i] ^= b.v[i];
118        }
119        return *this;
120    }
121    bitvector operator &(const bitvector &b) {
122        return bitvector(*this) &= b;
123    }
124    bitvector operator |(const bitvector &b) {
125        return bitvector(*this) |= b;
126    }
127    bitvector operator ^(const bitvector &b) {
128        return bitvector(*this) ^= b;
129    }
130    bitvector operator ~() const {
131        bitvector ret(*this);
132        for (int i = 0, ed = int(v.size()); i < ed; ++i) {
133            ret.v[i] = ~ret.v[i];
134        }
135        return ret;
136    }
137    bitvector operator <<=(const int t) {
138        bitvector ret(n);
139        ull last = 0;
140        int high = t >> 6, low = t & 63;
141        for(int i = 0; i + high < int(v.size()); ++i) {
142            ret.v[i + high] = last | (v[i] << low);
143            if (low) last = v[i] >> (64 - low);
144        }
145        return (*this) = ret;
146    }
```



```

147     bitvector operator >>=(const int t) {
148         bitvector ret(n);
149         ull last = 0;
150         int high = t >> 6, low = t & 63;
151         for(int i = int(v.size() - 1); i > high - 1; --i) {
152             ret.v[i - high] = last | (v[i] >> low);
153             if (low) last = v[i] << (64 - low);
154         }
155         return (*this) = ret;
156     }
157     bitvector operator <<(const int t) {
158         return bitvector(*this) <<= t;
159     }
160     bitvector operator >>(const int t) {
161         return bitvector(*this) >>= t;
162     }
163     std::string to_string() {
164         std::string ans;
165         for (int i = 0; i < n; ++i) {
166             ans += '0' + (*this)[i];
167         }
168         return ans;
169     }
170 };

```

chothlly.hpp

```

1  template <typename DAT>
2  struct coler_seg {
3      int l, r;
4      mutable DAT val;
5      coler_seg(int a = -1, int b = -1, DAT c = 0) : l(a), r(b), val(c) {}
6      bool operator<(const coler_seg&a) const { return l < a.l; }
7  };
8  template <typename DAT = int>
9  struct Chtholly : std::set<coler_seg<DAT>> {
10     using iterator = typename std::set<coler_seg<DAT>>::iterator;
11     void add(int l, int r, DAT val) {
12         iterator itr = split(r + 1), itl = split(l);
13         for (iterator it = itl; it != itr; ++it) {
14             it->val += val;
15         }
16     }
17     void assign(int l, int r, DAT val){
18         iterator itr = split(r + 1), itl = split(l);
19         erase(itl, itr);
20         emplace(l, r, val);
21     }
22     ll kth(int l, int r, int rk) {
23         iterator itr = split(r + 1), itl = split(l);
24         vector<pair<ll, int>> v;
25         for (auto it = itl; it != itr; ++it) {
26             v.emplace_back(it->val, it->r - it->l + 1);
27         }
28         sort(v);
29         for (const auto &[val, sz] : v) {
30             if (rk <= sz) return val;
31             rk -= sz;
32         }
33         return inf<ll>;
34     }
35     ll quis(int l, int r, int T, int mod) {

```

```

36     iterator itr = split(r + 1), itl = split(l);
37     ll res = 0;
38     for (iterator it = itl; it != itr; ++it) {
39         res = (res + (it->r - it->l + 1ll) * ksm((it->val) % mod, T, mod)) % mod;
40     }
41     return res;
42 }
43
44 private:
45 ll ksm(int a, int b, int mod) {
46     ll res = 1;
47     while (b) {
48         if (b & 1) res = (res * a) % mod;
49         a = 1ll * a * a % mod;
50         b >>= 1;
51     }
52     return res % mod;
53 }
54 iterator split(int pos) {
55     iterator it = lower_bound(coler_seg<DAT>(pos));
56     if (it != this->end() and it->l == pos) return it;
57     coler_seg<DAT> tmp = *--it;
58     erase(it);
59     emplace(tmp.l, pos - 1, tmp.val);
60     return emplace(pos, tmp.r, tmp.val).first;
61 }
62 };

```

dsu.hpp

```

1  struct dsu{      //MeIoN的dsu
2  public:
3      dsu(int _n) : n(_n), comp(_n), fa(_n), sz(_n, 1) {
4          std::iota(fa.begin(), fa.end(), 0);
5      }
6      int operator[](int x) { return ff(x); }
7      int size(int x) { return sz[ff(x)]; }
8      bool merge(int x, int y) {
9          x = ff(x), y = ff(y);
10         if (x == y) return false;
11         if (sz[x] < sz[y]) std::swap(x, y);
12         --comp;
13         sz[x] += sz[y], sz[y] = 0; fa[y] = x;
14         return true;
15     }
16     void rebuild() {
17         std::iota(fa.begin(), fa.end(), 0);
18         fill(sz, 1);
19     }
20 private:
21     int n, comp;
22     std::vector<int> fa, sz;
23     int ff(int x) {
24         while (x != fa[x]) x = fa[x] = fa[fa[x]];
25         return x;
26     }
27 };

```

fenw.hpp

```
1  template <class T = ll>
2  struct Fenw {
3      int n;
4      T total;
5      vector<T> dat;
6      Fenw() {}
7      Fenw(int n) { build(n); }
8      template <typename F>
9      Fenw(int n, F f) {
10         build(n, f);
11     }
12     Fenw(const vector<T> &v) { build(v); }
13
14     void build(int m) {
15         n = m;
16         dat.assign(m, T(0));
17         total = T(0);
18     }
19     void build(const vector<T> &v) {
20         build(v.size(), [&](int i) -> T { return v[i]; });
21     }
22     template <typename F>
23     void build(int m, F f) {
24         n = m;
25         dat.clear();
26         dat.reserve(n);
27         total = T(0);
28         for (int i = 0; i < n; ++i) dat.emplace_back(f(i));
29         for (int i = 1; i < n + 1; ++i) {
30             int j = i + (i & -i);
31             if (j < n + 1) {
32                 dat[j - 1] += dat[i - 1];
33             }
34         }
35         total = pre_sum(m);
36     }
37
38     void add(int k, T x) {
39         total += x;
40         for (++k; k < n + 1; k += k & -k) {
41             dat[k - 1] += x;
42         }
43     }
44
45     T sum_all() { return total; }
46     T prod(int k) { return pre_sum(k); }
47     T pre_sum(int k) {
48         chmin(k, n);
49         T res(0);
50         for (; k > 0; k -= k & -k) {
51             res += dat[k - 1];
52         }
53         return res;
54     }
55     T prod(int l, int r) {
56         chmax(l, 0);
57         chmin(r, n);
58         if (l == 0) return pre_sum(r);
59         T pos = T(0), neg = T(0);
60         while (l < r) {
61             pos += dat[r - 1];
```

```
62         r -= r & -r;
63     }
64     while (r < 1) {
65         neg += dat[l - 1];
66         l -= 1 & -1;
67     }
68     return pos - neg;
69 }
70 vector<T> get_all() {
71     vector<T> res(n);
72     for (int i = 0; i < n; ++i) {
73         res[i] = prod(i, i + 1);
74     }
75     return res;
76 }
77 };
78 struct Fenw01 {
79     int N, n;
80     vector<ull> dat;
81     Fenw<int> bit;
82     Fenw01() {}
83     Fenw01(int n) { build(n); }
84
85     void build(int m) {
86         N = m;
87         n = ceil(N + 1, 64);
88         dat.assign(n, ull(0));
89         bit.build(n);
90     }
91
92     void add(int k, int x) {
93         if (x == 1) add(k);
94         if (x == -1) remove(k);
95     }
96
97     void add(int k) {
98         dat[k / 64] |= 1ull << (k % 64);
99         bit.add(k / 64, 1);
100    }
101    void remove(int k) {
102        dat[k / 64] &= ~(1ull << (k % 64));
103        bit.add(k / 64, -1);
104    }
105
106    int sum_all() { return bit.sum_all(); }
107    int pre_sum(int k) {
108        int ans = bit.prod(k / 64);
109        ans += popcount(dat[k / 64] & ((1ull << (k % 64)) - 1));
110        return ans;
111    }
112    int prod(int k) { return pre_sum(k); }
113    int prod(int l, int r) {
114        if (l == 0) return pre_sum(r);
115        int ans = 0;
116        ans -= popcount(dat[l / 64] & ((1ull << (l % 64)) - 1));
117        ans += popcount(dat[r / 64] & ((1ull << (r % 64)) - 1));
118        ans += bit.prod(l / 64, r / 64);
119        return ans;
120    }
121 };
```

hashmap.hpp

```
1  template <typename Val>
2  struct hash_map {
3      hash_map(uint n = 0) { build(n); }
4      void build(uint n) {
5          uint k = 8;
6          while (k < (n << 1)) k <= 1;
7          cap = k >> 1, msk = k - 1;
8          key.resize(k), val.resize(k), used.assign(k, 0);
9      }
10     void clear() {
11         used.assign(used.size(), 0);
12         cap = msk + 1 >> 1;
13     }
14     int size() { return used.size() / 2 - cap; }
15     int index(const ull &k) {
16         int i = 0;
17         for (i = hash(k); used[i] and key[i] != k; i = (i + 1) & msk) {}
18         return i;
19     }
20
21     Val& operator[] (const ull &k) {
22         if (cap == 0) extend();
23         int i = index(k);
24         if (not used[i]) { used[i] = 1, key[i] = k, val[i] = Val{}, --cap; }
25         return val[i];
26     }
27
28     Val get(const ull &k, Val default_value) {
29         int i = index(k);
30         return (used[i] ? val[i] : default_value);
31     }
32
33     bool count(const ull &k) {
34         int i = index(k);
35         return used[i] and key[i] == k;
36     }
37
38     // f(key, val);
39     template <typename F>
40     void enumerate_all(F f) {
41         for (int i = 0, ed = used.size(); i < ed; ++i) {
42             if (used[i]) f(key[i], val[i]);
43         }
44     }
45 private :
46     uint cap, msk;
47     vector<ull> key;
48     vector<Val> val;
49     vector<bool> used;
50
51     ull hash(ull x) {
52         static const ull FIXED_RANDOM = std::chrono::steady_clock::now().time_since_epoch().count();
53         x += FIXED_RANDOM;
54         x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
55         x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
56         return (x ^ (x >> 31)) & msk;
57     }
58
59     void extend() {
```

```
60     vector<pair<ull, Val>> dat;  
61     dat.reserve(used.size() / 2 - cap);  
62     for (int i = 0, ed = used.size(); i < ed; ++i) {  
63         if (used[i]) dat.emplace_back(key[i], val[i]);  
64     }  
65     build(dat.size() << 1);  
66     for (auto &[a, b] : dat) (*this)[a] = b;  
67 }  
68 };
```

heap.hpp

```
1  template <typename T>  
2  struct heap {  
3      priority_queue<T> p, q;  
4  
5      void push(const T &x) {  
6          if (!q.empty() and q.top() == x) {  
7              q.pop();  
8              while (!q.empty() and q.top() == p.top()) {  
9                  p.pop(), q.pop();  
10             }  
11         } else {  
12             p.push(x);  
13         }  
14     }  
15     template <typename... Args>  
16     void emplace(Args &&...args) {  
17         if (!q.empty() and q.top() == T {std::forward<Args>(args)...}) {  
18             q.pop();  
19             while (!q.empty() and q.top() == p.top()) {  
20                 p.pop(), q.pop();  
21             }  
22         } else {  
23             p.emplace(std::forward<Args>(args)...);  
24         }  
25     }  
26     void pop() {  
27         p.pop();  
28         while (!q.empty() and p.top() == q.top()) {  
29             p.pop(), q.pop();  
30         }  
31     }  
32     void pop(const T &x) {  
33         if (p.top() == x) {  
34             p.pop();  
35             while (!q.empty() and p.top() == q.top()) {  
36                 p.pop(), q.pop();  
37             }  
38         } else {  
39             q.push(x);  
40         }  
41     }  
42     T top() { return p.top(); }  
43     bool empty() { return p.empty(); }  
44 };
```

rollback_array.hpp

```
1  template <typename T>  
2  struct RollbackArray {
```

```

3     int N;
4     std::vector<T> dat;
5     std::vector<std::pair<int, T>> history;
6     RollbackArray(std::vector<T> x) : N(x.size()), dat(x) {}
7     template <typename F>
8     RollbackArray(int N, F f) : N(N) {
9         dat.reserve(N);
10        for (int i = 0; i < N; ++i) {
11            dat.emplace_back(f(i));
12        }
13    }
14    int time() {
15        return history.size();
16    }
17    void rollback(int t) {
18        for (int i = time() - 1; i >= t; --i) {
19            auto& [idx, v] = history[i]; dat[idx] = v;
20        }
21        history.resize(t);
22    }
23    T get(int idx) {
24        return dat[idx];
25    }
26    void set(int idx, T x) {
27        history.emplace_back(idx, dat[idx]);
28        dat[idx] = x;
29    }
30    std::vector<T> get_all() {
31        std::vector<T> res(N);
32        for (int i = 0; i < N; ++i) {
33            res[i] = get(i);
34        }
35        return res; }
36    T operator[](int idx) {
37        return dat[idx];
38    }
39 };

```

rollback_dsu.hpp

```

1  #include "rollback_array.hpp"
2  struct rollback_dsu {
3      RollbackArray<int> dat;
4      rollback_dsu(int n) : dat(std::vector<int>(n, -1)) {}
5      int operator[](int v) {
6          while (dat.get(v) >= 0) {
7              v = dat.get(v);
8          }
9          return v;
10     }
11     int size(int v) {
12         return -dat.get((*this)[v]);
13     }
14     int time() {
15         return dat.time();
16     }
17     void rollback(int t) {
18         dat.rollback(t);
19     }
20     bool merge(int a, int b) {
21         a = (*this)[a], b = (*this)[b];
22         if (a == b) {

```



```

23         return false;
24     }
25     if (dat.get(a) > dat.get(b)) {
26         std::swap(a, b);
27     }
28     dat.set(a, dat.get(a) + dat.get(b));
29     dat.set(b, a);
30     return true;
31 }
32 };

```

splay.hpp

```

1  constexpr int N = 1'000'000 + 10 << 2;
2  struct MeIoN_Splay {
3      int fa[N], ch[N][2], num[N], siz[N], val[N], cnt, root;
4      int newnode(int n) {
5          val[++cnt] = n;
6          ch[cnt][0] = ch[cnt][1] = 0;
7          num[cnt] = siz[cnt] = 1;
8          return cnt;
9      }
10     int dir(int x) { return ch[fa[x]][1] == x; }
11     void upd(int x) { siz[x] = num[x] + siz[ch[x][0]] + siz[ch[x][1]]; }
12     void rotate(int x) {
13         int d = dir(x), f = fa[x];
14         if ((ch[f][d] = ch[x][d ^ 1]) != 0) fa[ch[f][d]] = f;
15         if ((fa[x] = fa[f]) != 0) ch[fa[x]][dir(f)] = x;
16         else root = x;
17         upd(ch[x][d ^ 1] = f);
18         upd(fa[f] = x);
19     }
20     void splay(int x) {
21         for (int f; (f = fa[x]) != 0; rotate(x))
22             if (fa[f]) rotate(dir(f) == dir(x) ? f : x);
23     }
24     void insert(int x) {
25         if (root == 0) {
26             root = newnode(x);
27             return;
28         }
29         int o = root;
30         while (1) {
31             int d = (val[o] < x);
32             if (val[o] == x) {
33                 ++num[o];
34                 ++siz[o];
35                 break;
36             } if (ch[o][d] == 0) {
37                 ch[o][d] = newnode(x);
38                 fa[ch[o][d]] = o;
39                 o = ch[o][d];
40                 break;
41             } else {
42                 o = ch[o][d];
43             }
44         }
45         splay(o);
46         upd(o);
47     }
48     void find(int x) {
49         // find the max v in tree that v <= x

```

```

50     int o = root, res = 0;
51     while (o != 0) {
52         if (val[o] == x) res = o, o = 0;
53         else if (val[o] < x) res = o, o = ch[o][1];
54         else if (ch[o][0] == 0 and res == 0) res = o, o = 0;
55         else o = ch[o][0];
56     }
57     splay(res);
58 }
59 void del(int x) {
60     find(x);
61     if (num[root] > 1) {
62         --num[root];
63         upd(root);
64     } else if (ch[root][0] == 0) {
65         fa[root = ch[root][1]] = 0;
66     } else if (ch[root][1] == 0) {
67         fa[root = ch[root][0]] = 0;
68     } else {
69         int _tmp = root, o = ch[root][0];
70         while (ch[o][1] != 0) o = ch[o][1];
71         splay(o);
72         ch[o][1] = ch[_tmp][1];
73         upd(fa[ch[_tmp][1]] = o);
74     }
75 }
76 int kth(int x) {
77     int o = root;
78     while (1) {
79         if (x <= siz[ch[o][0]]) o = ch[o][0];
80         else if (x <= siz[ch[o][0]] + num[o]) return val[o];
81         else x -= siz[ch[o][0]] + num[o], o = ch[o][1];
82     }
83 }
84 } splay;
85 /*
86     int n, m, ans;
87     std::cin >> n >> m;
88     for (int i = 0, op, x; i < m; ++i) {
89         std::cin >> op >> x;
90         if (op == 1) {
91             // insert
92             g.insert(x);
93         } else if (op == 2) {
94             // del
95             g.del(x);
96         } else if (op == 3) {
97             // rank of x ( may no x
98             g.insert(x);
99             g.find(x);
100             ans = g.siz[g.ch[g.root][0]] + 1;
101             g.del(x);
102         } else if (op == 4) {
103             // the num ranks x
104             ans = g.kth(x);
105         } else if (op == 5) {
106             // the num lower than x;
107             g.find(x - 1);
108             ans = g.val[g.root];
109         } else {
110             // the num greater than x
111             g.find(x);

```

```

112         if (g.val[g.root] > x) {
113             ans = g.val[g.root];
114         } else {
115             int o = g.ch[g.root][1];
116             while (g.ch[o][0] != 0) o = g.ch[o][0];
117             ans = g.val[o];
118         }
119     }
120 }
121 std::cout << ans << '\n';
122 */

```

sqrt_tree.hpp

```

1  template <typename Monoid>
2  struct sqrt_tree {    // nLog^2 预处理 O1查询区间信息 满足结合律
3      using MX = Monoid;
4      using X = typename MX::value_type;
5
6      static constexpr int K = 3;
7      static constexpr uint SZ[] = {8, 64, 4096};
8      static constexpr uint MASK[] = {7, 63, 4095};
9
10     int N;
11     // 元となる静的な列
12     vector<X> A;
13     // 各階層に対して、ブロック先頭からある要素まで [s,i]
14     // 各階層に対して、ある要素からブロック末尾まで [i,t]
15     vector<vector<X>> PREF, SUFF;
16     // 各階層に対して、あるブロックからあるブロックまで
17     vector<vector<X>> BETWEEN;
18
19     sqrt_tree() {}
20     template <typename F>
21     sqrt_tree(int n, F f) {
22         build(n, f);
23     }
24     sqrt_tree(const vector<X>& v) {
25         build(v.size(), [&](int i) -> X { return v[i]; });
26     }
27
28     template <typename F>
29     void build(int n_, F f) {
30         N = n_;
31         assert(N <= (1 << 24));
32         A.reserve(N);
33         for (int i = 0; i < N; ++i) A.emplace_back(f(i));
34         // まず prefix, suffix の構築
35         PREF.assign(K, A), SUFF.assign(K, A);
36         for (int k = 0; k < K; ++k) {
37             for (int i = 0; i < N; ++i) {
38                 if (i & MASK[k]) PREF[k][i] = MX::op(PREF[k][i - 1], A[i]);
39             }
40             for (int i = N; --i; ) {
41                 if (i & MASK[k]) SUFF[k][i - 1] = MX::op(A[i - 1], SUFF[k][i]);
42             }
43         }
44         // between の構築
45         BETWEEN.resize(K);
46         for (int k = 0; k < K; ++k) {
47             // n : 全体の小ブロックの個数
48             auto get = [&](int i) -> X { return SUFF[k][SZ[k] * i]; };

```

```

49     int n = N / SZ[k];
50     int s = 0;
51     for (int r = 0; r < n; ++r) {
52         if (r % SZ[k] == 0) s = r;
53         BETWEEN[k].emplace_back(get(r));
54         for (int l = r - 1; l >= s; --l) {
55             BETWEEN[k].emplace_back(MX::op(get(l), BETWEEN[k].back()));
56         }
57     }
58 }
59 }
60
61 static constexpr int BIT_TO_LAYER[] = {0, 0, 0, 1, 1, 1, 2, 2, 2, 2, 2, 2,
62                                         3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3};
63
64 X prod(int L, int R) {
65     assert(0 <= L && L <= R && R <= N);
66     if (L == R) return MX::unit();
67     if (L + 1 == R) return A[L];
68     --R;
69     int k = BIT_TO_LAYER[topbit(L ^ R)];
70     if (k == 0) {
71         // 長さ SZ[0] のブロックにクエリが収まっている。愚直に。
72         X x = A[L];
73         for (int i = L + 1; i < R + 1; ++i) x = MX::op(x, A[i]);
74         return x;
75     }
76     --k;
77     // 同じ長さ SZ[k+1] のブロック内にある。違う SZ[k] ブロック内にある。
78     uint a = L / SZ[k], b = R / SZ[k];
79     assert(a < b);
80     X &x1 = SUFF[k][L], &x2 = PREF[k][R];
81     if (a + 1 == b) return MX::op(x1, x2);
82     ++a, --b;
83     // [a,b] 番目の SZ[k]-block の間を取得する
84     // BETWEEN のどこにデータが置いてあるか調べる
85     uint m = a / SZ[k];
86     a &= MASK[k], b &= MASK[k];
87     uint idx = m * (SZ[k] / 2) * (SZ[k] + 1);
88     idx += (b + 1) * (b + 2) / 2 - 1 - a;
89     return MX::op(x1, MX::op(BETWEEN[k][idx], x2));
90 }
91 };

```

st_table.hpp

```

1 namespace RMQ {
2     vector<int> lg(2);
3     template <typename T> struct maxtable {
4         vector<T> a;
5         vector<vector<T>> st;
6         static int x;
7         maxtable(const vector<T> &b):a(b) {
8             int n = a.size(), i, j, k, r;
9             while (lg.size() <= n) lg.emplace_back(lg[lg.size() >> 1] + 1);
10            st.assign(lg[n] + 1, vector<T>(n));
11            std::iota(st[0].begin(), st[0].end(), 0);
12            for (j = 1; j <= lg[n]; j++) {
13                r = n - (1 << j);
14                k = 1 << j - 1;
15                for (i = 0; i <= r; i++)
16                    st[j][i] = a[st[j-1][i]] < a[st[j-1][i+k]] ? st[j-1][i+k] : st[j-1][i];

```

```

17         }
18     }
19     T quis(int l, int r) const { // [l, r]
20         assert(0 <= l and l <= r and r < a.size());
21         int z = lg[r - l + 1];
22         return std::max(a[st[z][l]], a[st[z][r - (1 << z) + 1]]);
23     }
24     int rmp(int l, int r) const {
25         assert(0 <= l and l <= r and r < a.size());
26         int z = lg[r-l+1];
27         return a[st[z][l]] < a[st[z][r - (1 << z) + 1]] ? st[z][r - (1 << z) + 1] : st[z][l];
28     }
29 };
30 } using RMQ::maxtable;

```

rollback mo

```

1  NAME MeIoN_is_UMP45(){
2      int n;
3      std::cin >> n;
4      vector<int> a(n);
5      std::cin >> a;
6      auto b = a;
7      unique(b);
8      for (auto &x : a)
9          x = MEION::lower_bound(b, x) - b.begin();
10     const int sz = b.size();
11     std::cin >> m;
12     const int B = std::ceil(std::sqrt(n)); assert(B);
13     using aa = array<int, 3>;
14     vector<aa> q(m);
15     vector<vector<aa>> Q(B);
16     for (int i = 0; auto &[l, r, id] : q) {
17         std::cin >> l >> r, --l, --r, id = i++;
18     }
19     MEION::sort(q, [&](const aa &a, const aa &b){
20         if (a[0] / B == b[0] / B) return a[1] < b[1]; return a[0] < b[0];
21     });
22     vector<int> pl(sz), pr(sz);
23     auto quis = [&](int l, int r) {
24         int ret = 0;
25         for (int i = l; i <= r; ++i) {
26             if (~pl[a[i]]) MAX(ret, i - pl[a[i]]);
27             else pl[a[i]] = i;
28         }
29         for (int i = l; i <= r; ++i) pl[a[i]] = -1;
30         return ret;
31     };
32     vector<int> res(m);
33     vector<int> del;
34     del.reserve(n << 2);
35     int pla(-1);
36     for (int i = 0, ie = (n - 1) / B + 1; i <= ie; ++i) {
37         int maxr = std::min(i * B + B - 1, n - 1), nr = maxr - 1, ret = 0;
38         MEION::fill(pl, -1), MEION::fill(pr, -1);
39         while (pla + 1 < m and q[pla + 1][0] / B == i) {
40             ++pla;
41             const auto &[L, R, id] = q[pla];
42             if (R / B == i) {
43                 res[id] = quis(L, R);
44                 continue;
45             }

```

```

46         while (nr < R) {
47             ++nr;
48             pr[a[nr]] = nr;
49             if (~pl[a[nr]]) MAX(ret, pr[a[nr]] - pl[a[nr]]);
50             else pl[a[nr]] = nr;
51         }
52         int nl = maxr, ans = ret;
53         while (nl > L) {
54             --nl;
55             if (~pr[a[nl]]) {
56                 MAX(ans, pr[a[nl]] - nl);
57             } else {
58                 pr[a[nl]] = nl;
59                 del.emplace_back(a[nl]);
60             }
61         }
62         res[id] = ans;
63         while (del.size()) {
64             pr[del.back()] = -1; del.pop_back();
65         }
66     }
67 }
68 for (const int i : res) std::cout << i << '\n';
69 }

```

block

```

1  struct block {
2      int n, off;
3      int a[0721], b[0721];
4      block(const vector<int> &x, int l, int r)
5          : n(r - l), off(l) {
6          for (int i = l; i < r; ++i) {
7              a[i - l] = x[i];
8          }
9          memcpy(b, a, sizeof a);
10         radix_sort(n, b);
11     }
12     void upd(int pla, int x) {
13         pla -= off;
14         if (pla < 0 or pla > n - 1) return;
15         a[pla] = x;
16         memcpy(b, a, sizeof a);
17         radix_sort(n, b);
18     }
19     int quis(int l, int r, int L, int R) {
20         l -= off, r -= off;
21         chmax(l, 0);
22         chmin(r, n);
23         if (l > r - 1) return 0;
24         if (r - l == n) {
25             return int(std::lower_bound(b, b + n, R) -
26                 std::lower_bound(b, b + n, L));
27         } else {
28             int ans = 0;
29             for (int i = l; i < r; ++i) {
30                 ans += a[i] > L - 1 and a[i] < R;
31             }
32             return ans;
33         }
34     }

```

35 };

flow

max_flow.hpp

```

1  namespace FL {
2      using flowt = long long;
3      constexpr int M = 3000000, N = 40000 + 10;
4      int y[M], nxt[M],
5          gap[N], fst[N], c[N], pre[N], q[N], dis[N];
6      pair<int, int> e[M];
7      flowt f[M];
8      int S, T, tot, Tn;
9      void III(int s, int t, int tn) {
10         tot = 1;
11         assert(tn < N);
12         for (int i = 0, iE = tn; i < iE; ++i) fst[i] = 0;
13         S = s, T = t, Tn = tn;
14     }
15     void add(int u, int v, flowt c1, flowt c2 = 0) {
16         tot++, y[tot] = v, f[tot] = c1, nxt[tot] = fst[u], fst[u] = tot;
17         e[tot] = {u, v};
18         tot++, y[tot] = u, f[tot] = c2, nxt[tot] = fst[v], fst[v] = tot;
19         e[tot] = {v, u};
20     }
21     flowt sap() {
22         int u = S, t = 1;
23         flowt flow = 0;
24         for (int i = 0; i < Tn; ++i) c[i] = fst[i], dis[i] = Tn, gap[i] = 0;
25         q[0] = T, dis[T] = 0, pre[S] = 0;
26         for (int i = 0; i < t; ++i) {
27             int u = q[i];
28             for (int j = fst[u]; j; j = nxt[j])
29                 if (dis[y[j]] > dis[u] + 1 and f[j ^ 1])
30                     q[t++] = y[j], dis[y[j]] = dis[u] + 1;
31         }
32         for (int i = 0; i < Tn; ++i) gap[dis[i]]++;
33         while (dis[S] <= Tn) {
34             while (c[u] and (not f[c[u]] or dis[y[c[u]]] + 1 != dis[u]))
35                 c[u] = nxt[c[u]];
36             if (c[u]) {
37                 pre[y[c[u]]] = c[u] ^ 1;
38                 u = y[c[u]];
39                 if (u == T) {
40                     flowt minf = inf<flowt>;
41                     for (int p = pre[T]; p; p = pre[y[p]])
42                         minf = std::min(minf, f[p ^ 1]);
43                     for (int p = pre[T]; p; p = pre[y[p]])
44                         f[p ^ 1] -= minf, f[p] += minf;
45                     flow += minf, u = S;
46                 }
47             } else {
48                 if (not(--gap[dis[u]])) break;
49                 int mind = Tn;
50                 c[u] = fst[u];
51                 for (int j = fst[u]; j; j = nxt[j])
52                     if (f[j] and dis[y[j]] < mind) mind = dis[y[j]], c[u] = j;
53                 dis[u] = mind + 1;
54                 gap[dis[u]]++;
55                 if (u != S) u = y[pre[u]];
56             }
57         }

```

```

58         return flow;
59     }
60 }; // namespace FL

```

max_flow_min_cost.hpp

```

1  namespace internal {
2      template <class E>
3      struct csr {
4          std::vector<int> start;
5          std::vector<E> elist;
6          explicit csr(int n, const std::vector<std::pair<int, E>>& edges)
7              : start(n + 1), elist(edges.size()) {
8              for (auto e: edges) { start[e.first + 1]++; }
9              for (int i = 1; i <= n; i++) { start[i] += start[i - 1]; }
10             auto counter = start;
11             for (auto e: edges) { elist[counter[e.first]++] = e.second; }
12         }
13     };
14     template <class T>
15     struct simple_queue {
16         std::vector<T> payload;
17         int pos = 0;
18         void reserve(int n) { payload.reserve(n); }
19         int size() const { return int(payload.size()) - pos; }
20         bool empty() const { return pos == int(payload.size()); }
21         void push(const T& t) { payload.push_back(t); }
22         T& front() { return payload[pos]; }
23         void clear() {
24             payload.clear();
25             pos = 0;
26         }
27         void pop() { pos++; }
28     };
29 } // namespace internal
30 /*
31  · atcoder Library をすこし改変したもの
32  · DAG = true であれば、負辺 OK (1 回目の最短路を dp で行う)
33  ただし、頂点番号は toposort されていることを仮定している。
34  */
35 template <class Cap = int, class Cost = ll, bool DAG = false>
36 struct mcf_graph {
37 public:
38     mcf_graph() {}
39     explicit mcf_graph(int n) : _n(n) {}
40     // frm, to, cap, cost
41     int add(int frm, int to, Cap cap, Cost cost) {
42         assert(0 <= frm && frm < _n), assert(0 <= to && to < _n), assert(0 <= cap), assert(DAG || 0 <= cost);
43         if (DAG) assert(frm < to);
44         int m = int(_edges.size());
45         _edges.push_back({frm, to, cap, 0, cost});
46         return m;
47     }
48     void DBEUG() {
49         std::cout << "flow graph\n";
50         std::cout << "frm, to, cap, cost\n";
51         for (auto&& [frm, to, cap, flow, cost]: _edges) U(frm, ' ', to, ' ', cap, ' ', cost, '\n');
52     }
53     struct edge {
54         int frm, to;
55         Cap cap, flow;
56         Cost cost;

```

```

57     };
58     edge get_edge(int i) {
59         int m = int(_edges.size());
60         assert(0 <= i && i < m);
61         return _edges[i];
62     }
63     std::vector<edge> edges() { return _edges; }
64
65     // (流量, 費用)
66     std::pair<Cap, Cost> flow(int s, int t) {
67         return flow(s, t, std::numeric_limits<Cap>::max());
68     }
69     // (流量, 費用)
70     std::pair<Cap, Cost> flow(int s, int t, Cap flow_limit) {
71         return slope(s, t, flow_limit).back();
72     }
73     // 返回流量和费用之间的关系曲线
74     std::vector<std::pair<Cap, Cost>> slope(int s, int t) {
75         return slope(s, t, std::numeric_limits<Cap>::max());
76     }
77     std::vector<std::pair<Cap, Cost>> slope(int s, int t, Cap flow_limit) {
78         assert(0 <= s && s < _n, assert(0 <= t && t < _n, assert(s != t);
79         int m = int(_edges.size());
80         std::vector<int> edge_idx(m);
81         auto g = [&]() {
82             std::vector<int> degree(_n, redge_idx(m);
83             std::vector<std::pair<int, _edge>> elist;
84             elist.reserve(2 * m);
85             for (int i = 0; i < m; i++) {
86                 auto e = _edges[i];
87                 edge_idx[i] = degree[e.frm]++;
88                 redge_idx[i] = degree[e.to]++;
89                 elist.push_back({e.frm, {e.to, -1, e.cap - e.flow, e.cost}});
90                 elist.push_back({e.to, {e.frm, -1, e.flow, -e.cost}});
91             }
92             auto _g = internal::csr<_edge>(_n, elist);
93             for (int i = 0; i < m; i++) {
94                 auto e = _edges[i];
95                 edge_idx[i] += _g.start[e.frm];
96                 redge_idx[i] += _g.start[e.to];
97                 _g.elist[edge_idx[i]].rev = redge_idx[i];
98                 _g.elist[redge_idx[i]].rev = edge_idx[i];
99             }
100             return _g;
101         }();
102         auto result = slope(g, s, t, flow_limit);
103         for (int i = 0; i < m; i++) {
104             auto e = g.elist[edge_idx[i]];
105             _edges[i].flow = _edges[i].cap - e.cap;
106         }
107         return result;
108     }
109
110 private:
111     int _n;
112     std::vector<edge> _edges;
113     // inside edge
114     struct _edge {
115         int to, rev;
116         Cap cap;
117         Cost cost;
118     };

```

```

119     std::vector<std::pair<Cap, Cost>> slope(internal::csr<_edge>& g, int s, int t, Cap flow_limit) {
120         if (DAG) assert(s == 0 && t == _n - 1);
121         std::vector<std::pair<Cost, Cost>> dual_dist(_n);
122         std::vector<int> prev_e(_n);
123         std::vector<bool> vis(_n);
124         struct Q {
125             Cost key;
126             int to;
127             bool operator<(Q r) const { return key > r.key; }
128         };
129         std::vector<int> que_min;
130         std::vector<Q> que;
131         auto dual_ref = [&]() {
132             for (int i = 0; i < _n; i++) {
133                 dual_dist[i].second = std::numeric_limits<Cost>::max();
134             }
135             std::fill(vis.begin(), vis.end(), false);
136             que_min.clear();
137             que.clear();
138             size_t heap_r = 0;
139             dual_dist[s].second = 0;
140             que_min.push_back(s);
141             while (!que_min.empty() || !que.empty()) {
142                 int v;
143                 if (!que_min.empty()) {
144                     v = que_min.back();
145                     que_min.pop_back();
146                 } else {
147                     while (heap_r < que.size()) {
148                         heap_r++;
149                         std::push_heap(que.begin(), que.begin() + heap_r);
150                     }
151                     v = que.front().to;
152                     std::pop_heap(que.begin(), que.end());
153                     que.pop_back();
154                     heap_r--;
155                 }
156                 if (vis[v]) continue;
157                 vis[v] = true;
158                 if (v == t) break;
159                 Cost dual_v = dual_dist[v].first, dist_v = dual_dist[v].second;
160                 for (int i = g.start[v]; i < g.start[v + 1]; i++) {
161                     auto e = g.elist[i];
162                     if (!e.cap) continue;
163                     Cost cost = e.cost - dual_dist[e.to].first + dual_v;
164                     if (dual_dist[e.to].second > dist_v + cost) {
165                         Cost dist_to = dist_v + cost;
166                         dual_dist[e.to].second = dist_to;
167                         prev_e[e.to] = e.rev;
168                         if (dist_to == dist_v) {
169                             que_min.push_back(e.to);
170                         } else {
171                             que.push_back(Q{dist_to, e.to});
172                         }
173                     }
174                 }
175             }
176             if (!vis[t]) { return false; }
177
178             for (int v = 0; v < _n; v++) {
179                 if (!vis[v]) continue;
180                 dual_dist[v].first -= dual_dist[t].second - dual_dist[v].second;

```

```

181         }
182         return true;
183     };
184
185     auto dual_ref_dag = [&]() {
186         for (int i = 0; i < _n; i++) {
187             dual_dist[i].second = std::numeric_limits<Cost>::max();
188         }
189         dual_dist[s].second = 0;
190         std::fill(vis.begin(), vis.end(), false);
191         vis[0] = true;
192
193         for (int v = 0; v < _n; ++v) {
194             if (!vis[v]) continue;
195             Cost dual_v = dual_dist[v].first, dist_v = dual_dist[v].second;
196             for (int i = g.start[v]; i < g.start[v + 1]; i++) {
197                 auto e = g.elist[i];
198                 if (!e.cap) continue;
199                 Cost cost = e.cost - dual_dist[e.to].first + dual_v;
200                 if (dual_dist[e.to].second > dist_v + cost) {
201                     vis[e.to] = true;
202                     Cost dist_to = dist_v + cost;
203                     dual_dist[e.to].second = dist_to;
204                     prev_e[e.to] = e.rev;
205                 }
206             }
207         }
208         if (!vis[t]) { return false; }
209
210         for (int v = 0; v < _n; v++) {
211             if (!vis[v]) continue;
212             dual_dist[v].first -= dual_dist[t].second - dual_dist[v].second;
213         }
214         return true;
215     };
216
217     Cap flow = 0;
218     Cost cost = 0, prev_cost_per_flow = -1;
219     std::vector<std::pair<Cap, Cost>> result = {{Cap(0), Cost(0)}};
220     while (flow < flow_limit) {
221         if (DAG && flow == 0) {
222             if (!dual_ref_dag()) break;
223         } else {
224             if (!dual_ref()) break;
225         }
226         Cap c = flow_limit - flow;
227         for (int v = t; v != s; v = g.elist[prev_e[v]].to) {
228             c = std::min(c, g.elist[g.elist[prev_e[v]].rev].cap);
229         }
230         for (int v = t; v != s; v = g.elist[prev_e[v]].to) {
231             auto& e = g.elist[prev_e[v]];
232             e.cap += c;
233             g.elist[e.rev].cap -= c;
234         }
235         Cost d = -dual_dist[s].first;
236         flow += c;
237         cost += c * d;
238         if (prev_cost_per_flow == d) { result.pop_back(); }
239         result.push_back({flow, cost});
240         prev_cost_per_flow = d;
241     }
242     return result;

```

```
243     }  
244 };
```

geo

两圆面积覆盖

```
1  ld dis = length(p1 - p2);
2  ld al = 2.01 * std::acos((r * r + dis * dis - rr * rr) / (2.01 * r * dis));
3  ld R2 = 0.51 * al * r * r - 0.51 * r * r * sin(al);
4  ld beta =
5      2.01 * std::acos((rr * rr + dis * dis - r * r) / (2.01 * rr * dis));
6  ld R1 = 0.51 * beta * rr * rr - 0.51 * rr * rr * sin(beta);
7  ld R = R1 + R2;
8  std::cout << R;
```

正n角形面积

```
1  ll n;                // 角数
2  ll r;                // 外接圆面积
3  ld S(ll n, ll r) {  // 菱形小块S
4      ld S;
5      S = (r * r * std::sin(pi / n) * std::sin(pi / (2 * n))) /
6          (2 * std::sin(pi - pi * 3 / 2 / n));
7      return S;
8  }
9  ld SS(ll n, ll r) { // n角形面积
10     ld res = S(n, r) * n * 2;
11     return res;
12 }
```

正n锥体体积

```
1  ld V(ll n, ll a) {
2      ld res(0);
3      res = a * a * a * n / (12 * std::tan(pi / n)) *
4          std::sqrt(1 - 1 / (4 * std::sin(pi / n) * std::sin(pi / n)));
5      return res;
6  }
```

1-base.hpp

```
1  using RE = long double;
2  template <typename T = int>
3  struct point {
4      T x, y;
5      point() : x(0), y(0) {}
6
7      template <typename A, typename B>
8      point(A x, B y) : x(x), y(y) {}
9
10     template <typename A, typename B>
11     point(pair<A, B> p) : x(p.first), y(p.second) {}
12
13     point operator+=(const point p) {
14         x += p.x, y += p.y;
15         return *this;
16     }
17     point operator-=(const point p) {
18         x -= p.x, y -= p.y;
19         return *this;
20     }
21     point operator+(point p) const {
22         return {x + p.x, y + p.y};
23     }
```



```

23     }
24     point operator-(point p) const {
25         return {x - p.x, y - p.y};
26     }
27     bool operator==(point p) const {
28         return x == p.x and y == p.y;
29     }
30     bool operator!=(point p) const {
31         return x != p.x or y != p.y;
32     }
33     point operator-() const {
34         return {-x, -y};
35     }
36     point operator*(T t) const {
37         return {x * t, y * t};
38     }
39     point operator/(T t) const {
40         return {x / t, y / t};
41     }
42
43     bool operator<(point p) const {
44         if (x != p.x) return x < p.x;
45         return y < p.y;
46     }
47     bool operator>(point p) const {
48         if (x != p.x) return x > p.x;
49         return y > p.y;
50     }
51     T dot(const point &other) const {
52         return x * other.x + y * other.y;
53     }
54     T det(const point &other) const {
55         return x * other.y - y * other.x;
56     }
57     T square() const {
58         return x * x + y * y;
59     }
60
61     RE length() { return sqrtl(x * x + y * y); }
62     RE angle() { return std::atan2(y, x); }
63
64     point rotate(double theta) {
65         static_assert(not std::is_integral<T>::value);
66         RE c = std::cos(theta), s = std::sin(theta);
67         return point{c * x - s * y, s * x + c * y};
68     }
69     point rot90(bool ccw = 1) {
70         return (ccw ? point{-y, x} : point{y, -x});
71     }
72 };
73
74 template <typename T>
75 std::istream& operator>>(std::istream& is, point<T>& any) {
76     is >> any.x >> any.y;
77     return is;
78 }
79 template <typename T>
80 std::ostream& operator<<(std::ostream& os, const point<T>& any) {
81     os << any.x << ' ' << any.y;
82     return os;
83 }
84

```

```
85 // A -> B -> Cと進むときに, 左转为 +1, 右转为 -1。
86 template<typename T>
87 int ccw(point<T> a, point<T> b, point<T> c) {
88     T x = (b - a).det(c - a);
89     if (x > 0) return 1;
90     if (x < 0) return -1;
91     return 0;
92 }
93
94 template <typename REAL = long double, typename T, typename U>
95 REAL dist(point<T> a, point<U> b) {
96     REAL dx = REAL(a.x) - REAL(b.x);
97     REAL dy = REAL(a.y) - REAL(b.y);
98     return std::sqrt(dx * dx + dy * dy);
99 }
100
101 // ax+by+c
102 template <typename T>
103 struct line {
104     T a, b, c;
105     line(T a, T b, T c) : a(a), b(b), c(c) {}
106     line(point<T> A, point<T> B) {
107         a = A.y - B.y;
108         b = B.x - A.x;
109         c = A.x * B.y - A.y * B.x;
110     }
111     line(T x1, T y1, T x2, T y2) : line(point<T>(x1, y1), point<T>(x2, y2)) {}
112
113     template <typename U>
114     U eval(point<U> p) const {
115         return a * p.x + b * p.y + c;
116     }
117
118     template <typename U>
119     T eval(U x, U y) const {
120         return a + x * b + y * c;
121     }
122
123     void normalize() {
124         static_assert(std::is_same_v<T, int> or std::is_same_v<T, long long>);
125         T gcd = std::gcd(std::gcd(std::abs(a), std::abs(b)), std::abs(c));
126         a /= gcd, b /= gcd, c /= gcd;
127     }
128
129     bool parallel(line other) const {
130         return a * other.b - b * other.a == 0;
131     }
132     bool is_orthogonal(line other) const {
133         return a * other.a + b * other.b == 0;
134     }
135 };
136
137 template <typename T>
138 struct segment {
139     point<T> a, b;
140
141     segment(point<T> a, point<T> b) : a(a), b(b) {}
142     segment(T x1, T y1, T x2, T y2) : segment(point<T>(x1, y1), point<T>(x2, y2)) {}
143
144     bool contain(point<T> c) const {
145         T det = (c - a).det(b - a);
146         if (det != 0) return 0;
```

```

147         return (c - a).dot(b - a) >= 0 and (c - b).dot(a - b) >= 0;
148     }
149
150     line<T> to_line() { return line(a, b); }
151 };
152
153 template <typename REAL>
154 struct circle {
155     point<REAL> O;
156     REAL r;
157     circle() : O(0, 0), r(0) {}
158     circle(point<REAL> O, REAL r) : O(O), r(r) {}
159     circle(REAL x, REAL y, REAL r) : O(x, y), r(r) {}
160     template <typename T>
161     bool contain(point<T> p){
162         REAL dx = p.x - O.x, dy = p.y - O.y;
163         return dx * dx + dy * dy <= r * r;
164     }
165 };
166
167 // 反射
168 template <typename T, typename U>
169 point<RE> reflection(point<T> p, line<U> l) {
170     RE t = RE(l.eval(p)) / (l.a * l.a + l.b * l.b);
171     RE x = p.x - 2 * t * l.a;
172     RE y = p.y - 2 * t * l.b;
173     return point<RE>(x, y);
174 }
175
176 // 不平行假定
177 template <typename REAL = long double, typename T>
178 point<REAL> cross_point(const line<T> l1, const line<T> l2) {
179     T det = l1.a * l2.b - l1.b * l2.a;
180     assert(det != 0);
181     REAL x = -REAL(l1.c) * l2.b + REAL(l1.b) * l2.c;
182     REAL y = -REAL(l1.a) * l2.c + REAL(l1.c) * l2.a;
183     return point<REAL>(x / det, y / det);
184 }
185
186 template <typename REAL = long double, typename T>
187 point<REAL> line_x_line(const line<T> l1, const line<T> l2) {
188     return cross_point<REAL, T>(l1, l2);
189 }
190
191 // 0: 0交点
192 // 1: 1交点
193 // 2: 无数交点
194 template <typename T>
195 int count_cross(segment<T> s1, segment<T> s2, bool include_ends) {
196     static_assert(!std::is_floating_point<T>::value);
197     line<T> l1 = s1.to_line();
198     line<T> l2 = s2.to_line();
199     if (l1.parallel(l2)) {
200         if (l1.eval(s2.a) != 0) return 0;
201         // 4 点在同一直线上
202         T a1 = s1.a.x, b1 = s1.b.x;
203         T a2 = s2.a.x, b2 = s2.b.x;
204         if (a1 == b1) {
205             a1 = s1.a.y, b1 = s1.b.y;
206             a2 = s2.a.y, b2 = s2.b.y;
207         }
208         if (a1 > b1) std::swap(a1, b1);
209         if (a2 > b2) std::swap(a2, b2);

```

```

209     T a = std::max(a1, a2);
210     T b = std::min(b1, b2);
211     if (a < b) return 2;
212     if (a > b) return 0;
213     return (include_ends ? 1 : 0);
214 }
215 // 不平行場合
216 T a1 = l2.eval(s1.a), b1 = l2.eval(s1.b);
217 T a2 = l1.eval(s2.a), b2 = l1.eval(s2.b);
218 if (a1 > b1) std::swap(a1, b1);
219 if (a2 > b2) std::swap(a2, b2);
220 bool ok1 = 0, ok2 = 0;
221 if (include_ends) {
222     ok1 = ((a1 <= T(0)) and (T(0) <= b1));
223     ok2 = ((a2 <= T(0)) and (T(0) <= b2));
224 } else {
225     ok1 = ((a1 < T(0)) and (T(0) < b1));
226     ok2 = ((a2 < T(0)) and (T(0) < b2));
227 }
228 return (ok1 and ok2 ? 1 : 0);
229 }
230
231 template <typename REAL, typename T>
232 vector<point<REAL>> cross_point(const circle<T> C, const line<T> L) {
233     T a = L.a, b = L.b, c = L.a * (C.O.x) + L.b * (C.O.y) + L.c;
234     T r = C.r;
235     bool sw = 0;
236     if (std::abs(a) < std::abs(b)) {
237         std::swap(a, b);
238         sw = 1;
239     }
240     //  $ax + by + c = 0, x^2 + y^2 = r^2$ 
241     T D = 4 * c * c * b * b - 4 * (a * a + b * b) * (c * c - a * a * r * r);
242     if (D < 0) return {};
243     REAL sqD = sqrtl(D);
244     REAL y1 = (-2 * b * c + sqD) / (2 * (a * a + b * b));
245     REAL y2 = (-2 * b * c - sqD) / (2 * (a * a + b * b));
246     REAL x1 = (-b * y1 - c) / a;
247     REAL x2 = (-b * y2 - c) / a;
248     if (sw) std::swap(x1, y1), std::swap(x2, y2);
249     x1 += C.O.x, x2 += C.O.x;
250     y1 += C.O.y, y2 += C.O.y;
251     if (D == 0) {
252         return {point<REAL>(x1, y1)};
253     }
254     return {point<REAL>(x1, y1), point<REAL>(x2, y2)};
255 }
256
257 template <typename REAL, typename T>
258 std::tuple<bool, point<T>, point<T>> cross_point_circle(circle<T> C1,
259                                                         circle<T> C2) {
260     using P = point<T>;
261     P O {0, 0};
262     P A = C1.O, B = C2.O;
263     if (A == B) return {false, O, O};
264     T d = (B - A).length();
265     REAL cos_val = (C1.r * C1.r + d * d - C2.r * C2.r) / (2 * C1.r * d);
266     if (cos_val < -1 || 1 < cos_val) return {false, O, O};
267     REAL t = std::acos(cos_val);
268     REAL u = (B - A).angle();
269     P X = A + P {C1.r * std::cos(u + t), C1.r * std::sin(u + t)};
270     P Y = A + P {C1.r * std::cos(u - t), C1.r * std::sin(u - t)};

```

```
271     return {true, X, Y};
272 }
```

2-apolloian_circle.hpp

```
1  #include "1-base.hpp"
2
3  // https://codeforces.com/contest/2/problem/C
4  template <typename REAL, typename T>
5  circle<REAL> apollonian_circle(point<T> A, point<T> B, T a, T b) {
6      assert(a != b);
7      point<REAL> X = (A * b + B * a) / (a + b);
8      point<REAL> Y = (A * b - B * a) / (b - a);
9      point<REAL> O = (X + Y) / 2.L;
10     REAL r = dist<REAL>(X, O);
11     return circle<REAL>(O.x, O.y, r);
12 }
```

3-angle_sort.hpp

```
1  #include "1-base.hpp"
2
3  template <typename T>
4  int lower_or_upper(const point<T> &p) {
5      if (p.y != 0) return (p.y > 0 ? 1 : -1);
6      if (p.x > 0) return -1;
7      if (p.x < 0) return 1;
8      return 0;
9  }
10
11 template <typename T>
12 int angle_cmp_3(const point<T> &L, const point<T> &R) {
13     int a = lower_or_upper(L), b = lower_or_upper(R);
14     if (a != b) return (a < b ? -1 : +1);
15     T det = L.det(R);
16     if (det > 0) return -1;
17     if (det < 0) return 1;
18     return 0;
19 }
20
21 template <typename T>
22 vector<int> angle_sort(const vector<point<T>> &v) {
23     vector<int> rk(v.size());
24     std::iota(rk.begin(), rk.end(), 0);
25     sort(rk, [&](auto &L, auto &R) -> bool{
26         return (angle_cmp_3(v[L], v[R]) == -1);
27     });
28     return rk;
29 }
30
31 template <typename T>
32 vector<int> angle_sort(const vector<pair<T, T>> &v) {
33     vector<point<T>> tmp(v.size());
34     for (int i = 0, ed = v.size(); i < ed; ++i) {
35         tmp[i] = point<T>(v[i]);
36     }
37     return angle_sort(tmp);
38 }
```

4-closest_pair.hpp

```
1  #include "1-base.hpp"
2  #include "3-angle_sort.hpp"
3  #include "../random/random.hpp"
4  #include "../ds/hashmap.hpp"
5  #include "8-distance.hpp"
6
7  using MeIoN_random_hash::shuffle, MeIoN_random_hash::hash_pair;
8
9  template <typename T>
10 pair<int, int> closest_pair(vector<point<T>> points) {
11     int n = points.size();
12     assert(n > 1);
13     hash_map<int> ma(n);
14     vector<int> id(n);
15     std::iota(id.begin(), id.end(), 0);
16     shuffle(id);
17     points = rearrange(points, id);
18
19     auto square = [&] (int i, int j) -> T {
20         return (points[j] - points[i]).square();
21     };
22
23     T best = square(0, 1);
24     pair<int, int> res(0, 1);
25     T w = sqrtl(best);
26
27     vector<int> nxt(n, -1);
28
29     auto ins = [&] (int i) -> void {
30         ull k = hash_pair(pair{points[i].x / w, points[i].y / w});
31         nxt[i] = ma.get(k, -1);
32         ma[k] = i;
33     };
34
35     auto quis = [&] (int i) -> bool {
36         assert(w);
37         ll a = points[i].x / w;
38         ll b = points[i].y / w;
39         bool upd = false;
40         for (int dx = -1; dx < 2; ++dx) {
41             for (int dy = -1; dy < 2; ++dy) {
42                 ull k = hash_pair<ll>({a + dx, b + dy});
43                 int j = ma.get(k, -1);
44                 while (j != -1) {
45                     if (chmin(best, square(i, j))) {
46                         upd = true;
47                         res = {i, j};
48                         w = std::sqrt(best);
49                     }
50                     j = nxt[j];
51                 }
52             }
53         }
54         return upd;
55     };
56
57     if (best == T(0)) {
58         res.first = id[res.first], res.second = id[res.second];
59         return res;
60     }
61     ins(0), ins(1);
```

```

62     for (int i = 2; i < n; ++i) {
63         if (quis(i)) {
64             if (best == T(0)) break;
65             ma.build(n);
66             for (int k = 0; k < i; ++k) {
67                 ins(k);
68             }
69         }
70         ins(i);
71     }
72     res.first = id[res.first], res.second = id[res.second];
73     return res;
74 }
75
76 template <typename T>
77 pair<int, int> closest_pair2(vector<point<T>> points) {
78     using RE = long double;
79     const int n = points.size();
80     if (n == 1) return pair(0, 0);
81     ld rd = MeIoN_random_hash::rng(114514) % 360 * 0.114514;
82     ld SIN = std::cos(rd), COS = std::sin(rd);
83     vector<int> id(n);
84     for (int i = 0; i < n; ++i) id[i] = i;
85     sort(id, [&](auto &a, auto &b) -> bool {
86         return points[a].x * COS - points[a].y * SIN <
87             points[b].x * COS - points[b].y * SIN;
88     });
89     ld best = distance<RE>(points[id[0]], points[id[1]]);
90     pair<int, int> ans = pair(id[0], id[1]);
91     for (int i = 0; i < n; ++i) {
92         for (int k = 1; k < 6 and i + k < n; ++k) {
93             if (chmin(best, distance<RE>(points[id[i]], points[id[i + k]]))) {
94                 ans = pair(id[i], id[i + k]);
95             }
96         }
97     }
98     return ans;
99 }

```

5-hull.hpp

```

1  #include "1-base.hpp"
2  // https://qoj.ac/problem/218
3  template <typename T, bool allow_180 = false>
4  vector<int> convex_hull(vector<point<T>> &p, string mode = "full",
5                          bool sorted = false) {
6      assert(mode == "full" or mode == "lower" or mode == "upper");
7      int n = p.size();
8      if (n == 1) return {0};
9      if (n == 2) {
10         if (p[0] < p[1]) return {0, 1};
11         if (p[0] > p[1]) return {1, 0};
12         return {0};
13     }
14     vector<int> id(n);
15     if (sorted) {
16         std::iota(id.begin(), id.end(), 0);
17     } else {
18         id = argsort(p);
19     }
20     if constexpr (allow_180) {
21         for (int i = 0; i < n - 1; ++i) {

```

```

22     assert(p[i] != p[i + 1]);
23 }
24 }
25 auto check = [&](int i, int j, int k) -> bool {
26     T det = (p[j] - p[i]).det(p[k] - p[i]);
27     if constexpr (allow_180) {
28         return det >= 0;
29     }
30     return det > T(0);
31 };
32 auto cal = [&]() {
33     vector<int> res;
34     for (const auto &k : id) {
35         while (res.size() > 1) {
36             auto i = res.end()[-2];
37             auto j = res.end()[-1];
38             if (check(i, j, k)) break;
39             res.pop_back();
40         }
41         res.emplace_back(k);
42     }
43     return res;
44 };
45 vector<int> res;
46 if (mode == "full" or mode == "lower") {
47     vector<int> Q = cal();
48     res.insert(res.end(), Q.begin(), Q.end());
49 }
50 if (mode == "full" or mode == "upper") {
51     if (not res.empty()) res.pop_back();
52     rev(id);
53     vector<int> Q = cal();
54     res.insert(res.end(), Q.begin(), Q.end());
55 }
56 if (mode == "upper") rev(res);
57 while (res.size() > 1 and p[res[0]] == p[res.back()]) res.pop_back();
58 return res;
59 }

```

6-convex_polygon.hpp

```

1  #include "5-hull.hpp"
2
3  // https://codeforces.com/contest/1906/problem/D
4
5  template <typename T>
6  struct convex_polygon {
7      using P = point<T>;
8      int n;
9      vector<P> points;
10     T area2;
11
12     // 需要传入一个凸包
13     convex_polygon(vector<P> points_) : n((int)points_.size()), points(points_) {
14         assert(n > 2);
15         area2 = 0;
16         for (int i = 0; i < n; ++i) {
17             int j = nxt_idx(i), k = nxt_idx(j);
18             assert((points[j] - points[i]).det(points[k] - points[i]) >= 0);
19             area2 += points[i].det(points[j]);
20         }
21     }

```



```

22
23 // comp(i, k)
24 template <typename F>
25 int periodic_min_comp(F comp) {
26     int l = 0, m = n, r = n << 1;
27     while (true) {
28         if (r - l == 2) break;
29         int lpos = (l + m) >> 1, rpos = (m + r + 1) >> 1;
30         if (comp(lpos % n, m % n)) {
31             r = m, m = lpos;
32         } else if (comp(rpos % n, m % n)) {
33             l = m, m = rpos;
34         } else {
35             l = lpos, r = rpos;
36         }
37     }
38     return m % n;
39 }
40
41 int nxt_idx(int i) { return (i + 1 == n ? 0 : i + 1); }
42 int pre_idx(int i) { return (i == 0 ? n - 1 : i - 1); }
43
44 // 中: 1, 境界: 0, 外: -1. test した.
45 int side(P p) {
46     int l = 1, r = n - 1;
47     T a = (points[l] - points[0]).det(p - points[0]);
48     T b = (points[r] - points[0]).det(p - points[0]);
49     if (a < 0 or b > 0) return -1;
50     // 从点 0 看, 点 p 位于 [L, R] 的方向
51     while (r - l > 1) {
52         int m = l + r >> 1;
53         T c = (points[m] - points[0]).det(p - points[0]);
54         if (c < 0) {
55             r = m, b = c;
56         } else {
57             l = m, a = c;
58         }
59     }
60     T c = (points[r] - points[l]).det(p - points[l]);
61     T x = std::min({a, -b, c});
62     if (x < 0) return -1;
63     if (x > 0) return 1;
64     // on triangle p[0] p[L] p[R]
65     if (p == points[0]) return 0;
66     if (c != 0 and a == 0 and l != 1) return 1;
67     if (c != 0 and b == 0 and r != n - 1) return 1;
68     return 0;
69 }
70
71 // return {min, idx} 点积最小值 O(Log)
72 pair<T, int> min_dot(P p) {
73     int idx = periodic_min_comp([&](int i, int k) -> bool {
74         return points[i].dot(p) < points[k].dot(p);
75     });
76     return {points[idx].dot(p), idx};
77 }
78
79 // return {max, idx} 点积最大值 O(Log)
80 pair<T, int> max_dot(P p) {
81     int idx = periodic_min_comp([&](int i, int k) -> bool {
82         return points[i].dot(p) > points[k].dot(p);
83     });

```

```

84         return {points[idx].dot(p), idx};
85     }
86
87     // 计算从一个点  $p$  观看多边形时，可以看到的多边形的范围
88     // 该函数返回两个索引，表示可以看到的范围（考虑反向偏角）
89     pair<int, int> visible_range(P p) {
90         int a = periodic_min_comp([&](int i, int k) -> bool {
91             return ((points[i] - p).det(points[k] - p) < 0);
92         });
93         int b = periodic_min_comp([&](int i, int k) -> bool {
94             return ((points[i] - p).det(points[k] - p) > 0);
95         });
96         if ((p - points[a]).det(p - points[pre_idx(a)]) == T(0)) a = pre_idx(a);
97         if ((p - points[b]).det(p - points[nxt_idx(b)]) == T(0)) b = nxt_idx(b);
98         return {a, b};
99     }
100
101     // 线段是否与凸多边形相交
102     bool check_cross(P pa, P pb) {
103         for (int i = 0; i < 2; ++i) {
104             std::swap(pa, pb);
105             auto [a, b] = visible_range(pa);
106             if ((points[a] - pa).det(pb - pa) >= 0) return false;
107             if ((points[b] - pa).det(pb - pa) <= 0) return false;
108         }
109         return true;
110     }
111
112     vector<T> AREA;
113
114     //  $point[i, \dots, j]$  (inclusive) 面积 * 2
115     T area_between(int i, int k) {
116         assert(i <= k and k <= n + i);
117         if (k == i + n) return area2;
118         i %= n, k %= n;
119         if (i > k) k += n;
120         if (AREA.empty()) build_AREA();
121         return AREA[k] - AREA[i] + (points[k % n].det(points[i]));
122     }
123
124     void build_AREA() {
125         AREA.resize(n << 1);
126         for (int i = 0; i < n; ++i) {
127             AREA[n + i] = AREA[i] = points[i].det(points[nxt_idx(i)]);
128         }
129         AREA.insert(AREA.begin(), T(0));
130         for (int i = 0; i < n * 2; ++i) {
131             AREA[i + 1] += AREA[i];
132         }
133     }
134 };

```

7-points_in_triangles.hpp

```

1  #include "../ds/fenw.hpp"
2  #include "3-angle_sort.hpp"
3  #include "../random/random.hpp"
4
5  // 输入点群A和B (Point<LL>)
6  // query(i,j,k): 返回三角形  $A_i A_j A_k$  内的 BL 数量 (非负数)
7  // 预处理  $O(NM \log M)$ , 查询  $O(1)$ 
8  // https://codeforces.com/contest/13/problem/D

```

```

9  struct count_points_in_triangles {
10     using P = point<ll>;
11     static constexpr int limit = 1'000'000'000 + 10;
12     vector<P> A, B;
13     vector<int> new_idx;    // 从 O 看到的极角序
14     vector<int> points;    // A[i] 与 B[k] 的匹配
15     vector<vector<int>> seg; // 线段 A[i] A[j] 内的 B[k]
16     vector<vector<int>> tri; // OA[i]A[j] 中的 B[k] 的数量
17     count_points_in_triangles(const vector<P> &a, const vector<P> &b)
18         : A(a), B(b) {
19         for (const auto p : A)
20             assert(std::max(std::abs(p.x), std::abs(p.y)) < limit);
21         for (const auto p : B)
22             assert(std::max(std::abs(p.x), std::abs(p.y)) < limit);
23         build();
24     }
25
26     int count3(int i, int j, int k) {
27         i = new_idx[i], j = new_idx[j], k = new_idx[k];
28         if (i > j) std::swap(i, j);
29         if (j > k) std::swap(j, k);
30         if (i > j) std::swap(i, j);
31         assert(i < j + 1 and j < k + 1);
32         ll d = (A[j] - A[i]).det(A[k] - A[i]);
33         if (d == 0) return 0;
34         if (d > 0) {
35             return tri[i][j] + tri[j][k] - tri[i][k] - seg[i][k];
36         }
37         int x = tri[i][k] - tri[i][j] - tri[j][k];
38         return x - seg[i][j] - seg[j][k] - points[j];
39     }
40
41     int count2(int i, int j) {
42         i = new_idx[i], j = new_idx[j];
43         if (i > j) std::swap(i, j);
44         return seg[i][j];
45     }
46
47 private:
48     P take_origin() {
49         // 不要让 OAiAj 和 OAiBj 在同一直线上
50         // fail prob: at most N(N+M)/LIM
51         // return P {-limit, MeIoN_random_hash::rng_64(-limit, limit)};
52         return P {-limit, MeIoN_random_hash::rng(-limit, limit)};
53     }
54
55     void build() {
56         P O = take_origin();
57         for (auto &p : A) {
58             p = p - O;
59         }
60         for (auto &p : B) {
61             p = p - O;
62         }
63         int N = A.size(), M = B.size();
64         vector<int> id = angle_sort(A);
65         A = rearrange(A, id);
66         new_idx.resize(N);
67         for (int i = 0; i < N; ++i) {
68             new_idx[id[i]] = i;
69         }
70     }

```

```

71     id = angle_sort(B);
72     B = rearrange(B, id);
73
74     points.assign(N, 0);
75     seg.assign(N, vector<int>(N));
76     tri.assign(N, vector<int>(N));
77
78     // points
79     for (int i = 0; i < N; ++i) {
80         for (int k = 0; k < M; ++k) {
81             if (A[i] == B[k]) {
82                 ++points[i];
83             }
84         }
85     }
86     /*
87     ll binary_search(F check, ll ok, ll ng, bool check_ok = true) {
88         if (check_ok) assert(check(ok));
89         while (abs(ok - ng) > 1) {
90             auto x = (ng + ok) >> 1;
91             (check(x) ? ok : ng) = x;
92         }
93         return ok;
94     }
95     */
96     int m = 0;
97     for (int j = 0; j < N; ++j) {
98         while (m < M and A[j].det(B[m]) < 0) ++m;
99         vector<P> C(m);
100         for (int k = 0; k < m; ++k) {
101             C[k] = B[k] - A[j];
102         }
103         vector<int> id(m);
104         for (int i = 0; i < m; ++i) id[i] = i;
105         sort(id,
106             [&](auto &a, auto &b) -> bool { return C[a].det(C[b]) > 0; });
107         C = rearrange(C, id);
108         vector<int> rk(m);
109         for (int k = 0; k < m; ++k) {
110             rk[id[k]] = k;
111         }
112         Fenw01 bit(m);
113
114         int k = m;
115         for (int i = j; i--;) {
116             while (k > 0 and A[i].det(B[k - 1]) > 0) {
117                 bit.add(rk[--k], 1);
118             }
119             P p = A[i] - A[j];
120             int lb = binary_search(
121                 [&](int n) -> bool {
122                     return (n == 0 ? true : C[n - 1].det(p) > 0);
123                 }, 0, m + 1);
124             int ub = binary_search(
125                 [&](int n) -> bool {
126                     return (n == 0 ? true : C[n - 1].det(p) >= 0);
127                 }, 0, m + 1);
128             seg[i][j] += bit.sum(lb, ub), tri[i][j] += bit.sum(lb);
129         }
130     }
131 }

```

```
132 };
```

8-distance.hpp

```
1  #include "1-base.hpp"
2
3  template <typename REAL = ld, typename T, typename U>
4  REAL distance(point<T> S, point<U> P) {
5      REAL dx = P.x - S.x;
6      REAL dy = P.y - S.y;
7      return std::sqrt(dx * dx + dy * dy);
8  }
9
10 template <typename REAL = ld, typename T, typename U>
11 REAL distance(segment<T> S, point<U> P) {
12     point<T> A = S.a, B = S.b;
13     bool b1 = (B - A).dot(P - A) >= 0;
14     bool b2 = (A - B).dot(P - B) >= 0;
15     if (b1 and not b2) {
16         return distance<REAL, T, T>(B, P);
17     }
18     if (not b1 and b2) {
19         return distance<REAL, T, T>(A, P);
20     }
21     line<T> L = S.to_line();
22     return REAL(std::abs(L.eval(P))) / std::sqrt(REAL(L.a) * L.a + REAL(L.b) * L.b);
23 }
24
25 template <typename REAL, typename T>
26 REAL distance(segment<T> s1, segment<T> s2) {
27     if (count_cross<T>(s1, s2, true)) return REAL(0);
28     REAL res = distance<REAL, T, T>(s1, s2.a);
29     chmin(res, distance<REAL, T, T>(s1, s2.b));
30     chmin(res, distance<REAL, T, T>(s2, s1.a));
31     chmin(res, distance<REAL, T, T>(s2, s1.b));
32     return res;
33 }
```

9-furthest_pair.hpp

```
1  #include "1-base.hpp"
2  #include "5-hull.hpp"
3  #include "4-closest_pair.hpp"
4  #include "8-distance.hpp"
5
6  // https://www.luogu.com.cn/problem/P6247
7  template <typename T>
8  pair<int, int> furthest_pair(vector<point<T>> points) {
9      T best = -1;
10     pair<int, int> ans = {-1, -1};
11
12     auto upd = [&](int i, int j) -> void {
13         point<T> p = points[i] - points[j];
14         ll d = p.dot(p);
15         if (chmax(best, d)) ans = pair(i, j);
16     };
17     upd(0, 1);
18
19     vector<int> id = convex_hull(points);
20     int n = id.size();
21     if (n == 1) return ans;
```

```

22     if (n == 2) return pair(id[0], id[1]);
23     /*
24     用两条与直径垂直的平行线夹住凸包
25     两条平行线夹住的两点是候补点
26     将 $p[i]p[i+1]$ 的相对侧设为候选即可
27     */
28     for (int i = 0; i < n; ++i) {
29         id.emplace_back(id[i]);
30     }
31
32     vector<point<T>> C = rearrange(points, id);
33     int j = 1;
34     for (int i = 0; i < n; ++i) {
35         chmax(j, i);
36         while (j < 2 * n and (C[i + 1] - C[i]).det(C[j + 1] - C[j]) > 0) ++j;
37         upd(id[i], id[j]);
38     }
39     return ans;
40 }

```

10-triangle_area.hpp

```

1  #include "1-base.hpp"
2
3  template <typename RE = long double, typename T>
4  RE triangle_area(point<T> a, point<T> b, point<T> c) {
5      return std::abs((b - a).det(c - a)) * 0.5L;
6  }
7  template <typename RE = ll, typename T = ll>
8  RE triangle_area_2(point<T> a, point<T> b, point<T> c) {
9      return std::abs((b - a).det(c - a));
10 }

```

11-in_circle.hpp

```

1  #include "1-base.hpp"
2  #include "10-triangle_area.hpp"
3
4  template <typename REAL, typename T>
5  circle<REAL> in_circle(point<T> A, point<T> B, point<T> C) { // 内接圆
6      REAL a = distance<REAL, T, T>(B, C);
7      REAL b = distance<REAL, T, T>(C, A);
8      REAL c = distance<REAL, T, T>(A, B);
9      REAL x = (a * A.x + b * B.x + c * C.x) / (a + b + c);
10     REAL y = (a * A.y + b * B.y + c * C.y) / (a + b + c);
11     REAL r = 2 * triangle_area<REAL>(A, B, C) / (a + b + c);
12     return Circle<REAL>(x, y, r);
13 }

```

12-line_inside_polygon.hpp

```

1  #include "1-base.hpp"
2  template <typename T>
3  bool inside_polygon(const vector<point<T>> &polygon, segment<T> s) { // 判断线段是否在多边形内部
4      using P = Point<T>;
5      int n = polygon.size();
6      int cnt = 0;
7      P A = s.A, B = s.B;
8      auto prev = [&](int i) -> int { return i == 0 ? n - 1 : i - 1; };
9      auto next = [&](int i) -> int { return i == n - 1 ? 0 : i + 1; };
10     for (int i = 0; i < n; ++i) {

```

```

11     P p = polygon[i], q = polygon[next(i)], r = polygon[prev(i)];
12     int a = ccw(A, B, p);
13     int b = ccw(A, B, q);
14     int c = ccw(A, B, r);
15     if (a * b == -1) {
16         segment pq(p, q);
17         auto L = pq.to_Line();
18         T x = L.eval(A), y = L.eval(B);
19         if (x < y) {
20             x = -x, y = -y;
21         }
22         if (x <= 0) {
23             ++cnt;
24         }
25         if (0 < x and x < x - y) {
26             return false;
27         }
28     }
29     if (a == 0) {
30         if (b != c and (b * c < 0 or ccw(p, q, r) > 0)) {
31             T t = (p - a).dot(B - A), x = (B - A).dot(B - A);
32             if (t <= 0) ++cnt;
33             if (0 < t and t < x) {
34                 return false;
35             }
36         }
37     }
38     return (cnt % 2 == 1);
39 }
40 }

```

13-manhattan_mst.hpp

```

1  #include "1-base.hpp"
2  #include "../ds/dsu.hpp"
3
4  template <typename T>
5  vector<vector<pair<int, T>>> manhattan_mst(vector<point<T>> &points) {
6      int n = points.size();
7      vector<std::tuple<T, int, int>> dat;
8      dat.reserve(n << 2);
9      vector<int> rk(n);
10     std::iota(rk.begin(), rk.end(), 0);
11
12     for (int a = 0; a < 2; ++a) {
13         for (auto && [ x, y ] : points) {
14             x = -x;
15         }
16         for (int b = 0; b < 2; ++b) {
17             for (auto && [ x, y ] : points) {
18                 std::swap(x, y);
19             }
20             sort(rk, [&](const int &i, const int &j) -> bool {
21                 return points[i].x + points[i].y <
22                     points[j].x + points[j].y;
23             });
24
25             map<T, int> mp;
26             for (const int i : rk) {
27                 auto & [ x, y ] = points[i];
28                 for (auto it = mp.lower_bound(-y); it != mp.end();
29                     it = mp.erase(it)) {

```

```

30         const int j = it->second;
31         auto &[xj, yj] = points[j];
32         const int dx = x - xj;
33         const int dy = y - yj;
34         if (dy > dx) break;
35         dat.emplace_back(dx + dy, i, j);
36     }
37     mp[-y] = i;
38 }
39 }
40 }
41
42 sort(dat);
43 dsu g(n);
44 vector<vector<pair<int, T>>> v(n);
45 for (auto &[cost, i, j] : dat) {
46     if (g.merge(i, j)) {
47         v[i].emplace_back(j, cost);
48         v[j].emplace_back(i, cost);
49     }
50 }
51 return v;
52 }

```

14-max_norm_sum.hpp

```

1  #include "1-base.hpp"
2  #include "3-angle_sort.hpp"
3
4  template <typename VAL, typename T>
5  VAL max_norm_sum(const vector<point<T>> &points) { // 一堆向量选一部分最大模长
6      const int n = points.size();
7      vector<point<T>> v(points);
8      point<T> c{0, 0};
9      for (const auto &[x, y] : points) {
10         if (y > 0 or (y == 0 and x < 0)) {
11             c.x += x, c.y += y;
12         }
13         v.emplace_back(-x, -y);
14     }
15     vector<int> rk = angle_sort(v);
16     v = rearrange(v, rk);
17
18     auto eval = [&]() -> VAL {
19         return VAL(c.x) * c.x + VAL(c.y) * c.y;
20     };
21
22     VAL ans = eval();
23     for (int i = 0; i < (n << 1); ++i) {
24         c = c + v[i];
25         chmax(ans, eval());
26     }
27     return ans;
28 }

```

15-minkowski_sum.hpp

```

1  #include "1-base.hpp"
2  #include "3-angle_sort.hpp"
3  #include "5-hull.hpp"
4  #include "6-convex_polygon.hpp"

```



```

5
6  template <typename T>
7  vector<point<T>> minkowski_sum(vector<point<T>> A,
8                                vector<point<T>> B) {
9      using P = point<T>;
10     vector<P> F;
11     P p(0, 0);
12     for (int i = 0; i < 2; ++i) {
13         std::swap(A, B);
14         vector<P> points = A;
15         int n = points.size();
16         for (int i = 0; i < n; ++i) {
17             int k = (i + 1) % n;
18             F.emplace_back(points[k] - points[i]);
19         }
20         p = p + *min_element(points.begin(), points.end());
21     }
22     auto rk = angle_sort(F);
23     int n = rk.size();
24     F = rearrange(F, rk);
25     vector<P> points(n);
26     for (int i = 0; i < n - 1; ++i) {
27         points[i + 1] = points[i] + F[i];
28     }
29     P add = p - *min_element(points.begin(), points.end());
30     for (auto &x : points) {
31         x += add;
32     }
33     rk = convex_hull(points);
34     points = rearrange(points, rk);
35     return points;
36 }

```

16-out_circle.hpp

```

1  #include "1-base.hpp"
2
3  template <typename RE, typename T>
4  circle<RE> out_circle(point<T> A, point<T> B, point<T> C) {
5      RE b1 = B.x - A.x, b2 = B.y - A.y;
6      RE c1 = C.x - A.x, c2 = C.y - A.y;
7      RE bb = (b1 * b1 + b2 * b2) / 2;
8      RE cc = (c1 * c1 + c2 * c2) / 2;
9
10     RE det = b1 * c2 - b2 * c1;
11     RE x = (bb * c2 - b2 * cc) / det;
12     RE y = (b1 * cc - bb * c1) / det;
13     RE r = std::sqrt(x * x + y * y);
14     x += A.x, y += A.y;
15     return circle<RE>(x, y, r);
16 }
17
18 template <typename T>
19 int out_circle_side(point<T> A, point<T> B, point<T> C, point<T> p) {
20     T d = (B - A).det(C - A);
21     assert(d != 0);
22     if (d < 0) std::swap(B, C);
23     array<point<T>, 3> pts = {A, B, C};
24     array<array<T, 3>, 3> mat;
25     for (int i = 0; i < 3; ++i) {
26         T dx = pts[i].x - p.x, dy = pts[i].y - p.y;
27         mat[i][0] = dx, mat[i][1] = dy, mat[i][2] = dx * dx + dy * dy;

```

```

28     }
29     T det = 0;
30     det += mat[0][0] * (mat[1][1] * mat[2][2] - mat[1][2] * mat[2][1]);
31     det += mat[0][1] * (mat[1][2] * mat[2][0] - mat[1][0] * mat[2][2]);
32     det += mat[0][2] * (mat[1][0] * mat[2][1] - mat[1][1] * mat[2][0]);
33     if (det == 0) return 0;
34     return (det > 0 ? 1 : -1);
35 }

```

17-minimum_enclosing_circle.hpp

```

1  #include "1-base.hpp"
2  #include "16-out_circle.hpp"
3  #include "3-angle_sort.hpp"
4  #include "5-hull.hpp"
5  #include "15-minkowski_sum.hpp"
6
7  template <typename RE, typename T>
8  std::tuple<circle<RE>, int, int, int> minimum_enclosing_circle( // 一组点的最小包围圆 (某三个点的外接圆
9      vector<point<T>> points) {
10     const int n = points.size();
11     assert(n != 0);
12     if (n == 1) {
13         circle<RE> C(points[0].x, points[0].y, 0);
14         return {C, 0, -1, -1};
15     }
16     vector<int> rk(n);
17     std::iota(rk.begin(), rk.end(), 0);
18     for (int i = 0, k; i < n; ++i) {
19         k = rng() % (i + 1);
20         if (i != k) {
21             std::swap(rk[i], rk[k]);
22         }
23     }
24
25     points = rearrange(points, rk);
26
27     std::tuple<int, int, int> c = {0, -1, -1};
28     auto contain = [&](point<T> p) -> bool {
29         auto [i, k, j] = c;
30         if (k == -1) {
31             return p == points[i];
32         }
33         if (j == -1) {
34             return (points[i] - p).dot(points[k] - p) <= 0;
35         }
36         return out_circle_side(points[i], points[k], points[j], p) >= 0;
37     };
38     for (int i = 1; i < n; ++i) {
39         if (contain(points[i])) continue;
40         c = {0, i, -1};
41         for (int k = 1; k < i; ++k) {
42             if (contain(points[k])) continue;
43             c = {i, k, -1};
44             for (int j = 0; j < k; ++j) {
45                 if (contain(points[j])) continue;
46                 c = {i, k, j};
47             }
48         }
49     }
50     auto [i, k, j] = c;
51     if (j == -1) {

```

```
52     RE x1 = points[i].x;
53     RE y1 = points[i].y;
54     RE x2 = points[k].x;
55     RE y2 = points[k].y;
56     point<RE> O = {0.5 * (x1 + x2), 0.5 * (y1 + y2)};
57     RE r = sqrtl((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2)) / 2;
58     circle<RE> C(0, r);
59     return {C, rk[i], rk[k], -1};
60 }
61 circle<RE> C = out_circle<RE>(points[i], points[k], points[j]);
62 return {C, rk[i], rk[k], rk[j]};
63 }
```

graph

2_sat.hpp

```
1  struct TwoSat { // MeIoNの2-sat
2  private:
3      int n, tot, cnt;
4      vector<vector<int>> v;
5      vector<bool> ans, vis;
6      vector<int> dfn, low, id, s;
7      void add(int x, int y) {
8          v[x].emplace_back(y);
9          v[y ^ 1].emplace_back(x ^ 1);
10     }
11     void tarjan(int n) {
12         dfn[n] = low[n] = ++tot;
13         vis[n] = 1;
14         s.emplace_back(n);
15         for (const int i : v[n]) {
16             if (not dfn[i]) {
17                 tarjan(i);
18                 chmin(low[n], low[i]);
19             } else if (vis[i]) {
20                 chmin(low[n], dfn[i]);
21             }
22         }
23         if (dfn[n] == low[n]) {
24             while (1) {
25                 int i = s.back();
26                 s.pop_back();
27                 id[i] = cnt, vis[i] = 0;
28                 if (i == n) break;
29             } ++cnt;
30         }
31     }
32 public:
33     TwoSat(int n) : n(n), v(n << 1), ans(n), dfn(n << 1), low(n << 1), vis(n << 1), id(n << 1) {}
34     void ban(int x, int y, int val_X, int val_Y) {
35         val_X ^= 1;
36         add(x << 1 | val_X, y << 1 | val_Y);
37     }
38     void either(int x, int y, int val_X, int val_Y) {
39         ban(x, y, not val_X, not val_Y);
40     }
41     void either(int x, int y) { ban(x, y, 0, 0); }
42     void to(int x, int y) { ban(x, y, 1, 0); }
43     void both(int x, int y) {
44         ban(x, y, 0, 0);
45         ban(x, y, 0, 1);
46         ban(x, y, 1, 0);
47     }
48     bool solve() {
49         s.clear();
50         std::fill(dfn.begin(), dfn.end(), 0);
51         std::fill(low.begin(), low.end(), 0);
52         std::fill(vis.begin(), vis.end(), false);
53         tot = 0, cnt = 0;
54         for (int i = 0; i < n << 1; ++i) if (not dfn[i]) tarjan(i);
55         for (int i = 0; i < n; ++i) {
56             if (id[i << 1] == id[i << 1 | 1]) return false;
57             if (id[i << 1] < id[i << 1 | 1]) {
```

```
58         ans[i] = 1;
59     }
60 }
61 return true;
62 }
63 vector<bool> answer() { return ans; }
64 };
```

dijkstra.hpp

```
1  template <typename T, typename VAL>
2  pair<vector<T>, vector<int>> dijkstra(const vector<vector<pair<int, VAL>>> &v, int s) {
3      const int n = v.size();
4      vector<T> dis(n, inf<T>);
5      vector<int> fa(n, -1);
6
7      using P = pair<T, int>;
8      priority_queue<P, vector<P>, greater<P>> q;
9
10     dis[s] = 0;
11     q.emplace(0, s);
12     while (not q.empty()) {
13         auto [dv, n] = q.top();
14         q.pop();
15         if (dv > dis[n]) continue;
16         for (auto [i, w] : v[n]) {
17             if (chmin(dis[i], dis[n] + w)) {
18                 fa[i] = n;
19                 q.emplace(dis[i], i);
20             }
21         }
22     }
23     return {dis, fa};
24 }
```

三元环计数

```
1  NAME MeIoN_is_UMP45() {
2      int n, m;
3      std::cin >> n >> m;
4      vector<vector<int>> v(n);
5      vector<int> d(n);
6      vector<pair<int, int>> e;
7      for (int i = 0, l, r; i < m; ++i) {
8          std::cin >> l >> r, --l, --r;
9          ++d[l], ++d[r];
10         e.emplace_back(l, r);
11     }
12     for (const auto &[l, r] : e) {
13         if (d[l] < d[r]) {
14             v[l].emplace_back(r);
15         } else if (d[l] > d[r]) {
16             v[r].emplace_back(l);
17         } else {
18             v[std::min(l, r)].emplace_back(std::max(l, r));
19         }
20     }
21     ll ans = 0;
22     vector<bool> tag(n);
23     for (int i = 0; i < n; ++i) {
24         for (auto k : v[i]) {
```

```
25         tag[k] = true;
26     }
27     for (auto k : v[i]) {
28         for (auto j : v[k]) {
29             if (tag[j]) {
30                 ++ans;
31             }
32         }
33     }
34     for (auto k : v[i]) {
35         tag[k] = false;
36     }
37 }
38 std::cout << ans << '\n';
39 }
```

最大团

```
1  ll n, ans, anss;
2
3  namespace max_t_t{
4      vector<vector<int>>> v;
5      vector<int> cnt, vis, res;
6      int dfs(int x,int now){
7          for (int i = x + 1, iE = n; i <= iE; ++i) {
8              if (cnt[i] + now <= ans) return 0;
9              if (not v[x][i]) continue;
10             int k;
11             for (k = 1; k < now;k++){
12                 if (not v[i][vis[k]]) break;
13             }
14             if (k == now){
15                 vis[now] = i;
16                 if (dfs(i, now + 1))
17                     return 1;
18             }
19         }
20         if (now > ans + 1){
21             ans = now - 1;
22             for (int i = 1; i < ans; ++i) {
23                 res[i] = vis[i];
24             }
25             return 1;
26         }
27         return 0;
28     }
29     void work(){
30         ans = -1;
31         for (int i = n; i; i--){
32             vis[1] = i;
33             dfs(i, 2);
34             cnt[i] = ans;
35         }
36     }
37     void build(int n){
38         v.assign(n + 1, vector<int>(n + 1, 0));
39         cnt = vector<int>(n + 1, 0);
40         vis = res = cnt;
41         for (int i = 1, iE = n - 1; i <= iE; ++i) {
42             int x, y;
43             std::cin >> x >> y;
44             v[x][y] = v[y][x] = 1;
```

```
45     }
46 }
47 }using namespace max_t_t;
48 int main(){
49     int n;
50     build(n);
51     work();
52     std::cout << ans << '\n';
53     for (int i = 1; i < ans; ++i) {
54         std::cout << res[i] << " ";
55     }
56     return 0;
57 }
```

math

exgcd.hpp

```
1 ll exgcd(ll a, ll b, ll &x, ll &y){
2     if (b == 0){
3         x = 1, y = 0;
4         return a;
5     }
6     ll d = exgcd(b, a % b, y, x);
7     y -= a / b * x;
8     return d;
9 }
```

mat.hpp

```
1 template <class mint, ull n>
2 struct MT : array<array<mint, n>, n> {
3     MT(int x = 0, int y = 0) {
4         for (int i = 0; i < n; ++i) {
5             for (int k = 0; k < n; ++k) {
6                 (*this)[i][k] = y;
7             }
8         }
9         for (int i = 0; i < n; ++i) {
10             (*this)[i][i] = x;
11         }
12     }
13     template <typename T, ull N>
14     MT(array<array<T, N>, N> &base) {
15         assert(N <= n);
16         for (int i = 0; i < N; ++i) {
17             for (int k = 0; k < N; ++k) {
18                 (*this)[i][k] = base[i][k];
19             }
20         }
21     }
22     template <typename T> MT(vector<vector<T>>& base) {
23         assert(base.size() <= n and base[0].size() <= n);
24         for (int i = 0; i < base.size(); ++i) {
25             for (int k = 0; k < base[0].size(); ++k) {
26                 (*this)[i][k] = base[i][k];
27             }
28         }
29     }
30     MT& operator*=(const MT& p) {
31         MT res;
32         for (int i = 0; i < n; ++i) {
33             for (int j = 0; j < n; ++j) {
34                 for (int k = 0; k < n; ++k) {
35                     res[i][j] += (*this)[i][k] * p[k][j];
36                 }
37             }
38         }
39         return *this = res;
40     }
41     MT operator*(const MT& p) {
42         return MT(*this) *= p;
43     }
44     MT ksm(int k, bool ok = false) {
45         MT res(1);
```



```

46     for (; k; k >>= 1) {
47         if (k & 1) {
48             res *= (*this);
49         }
50         (*this) *= (*this);
51     }
52     if (ok) {
53         (*this) = res;
54     }
55     return res;
56 }
57 MT ksm(ll k, bool ok = false) {
58     MT res(1);
59     for (; k; k >>= 1) {
60         if (k & 1) {
61             res *= (*this);
62         }
63         (*this) *= (*this);
64     }
65     if (ok) {
66         (*this) = res;
67     }
68     return res;
69 }
70 };

```

prims_set.hpp

```

1  struct m64 {
2      using i64 = long long;
3      using u64 = unsigned long long;
4      inline static u64 m, r, n2; // r * m = -1 (mod 1<<64), n2 = 1<<128 (mod m)
5      static void set_mod(u64 m) {
6          assert(m < (1ull << 62));
7          assert((m & 1) == 1);
8          m64::m = m;
9          n2 = -u128(m) % m;
10         r = m;
11         for (ll _ = 0; _ < ll(5); ++_) r *= 2 - m * r;
12         r = -r;
13         assert(r * m == -1ull);
14     }
15     static u64 reduce(u128 b) { return (b + u128(u64(b) * r) * m) >> 64; }
16     u64 x;
17     m64() : x(0) {}
18     m64(u64 x) : x(reduce(u128(x) * n2)){};
19     u64 val() const { u64 y = reduce(x); return y >= m ? y - m : y; }
20     m64 &operator+=(m64 y) { x += y.x - (m << 1); x = (i64(x) < 0 ? x + (m << 1) : x); return *this; }
21     m64 &operator-=(m64 y) { x -= y.x; x = (i64(x) < 0 ? x + (m << 1) : x); return *this; }
22     m64 &operator*=(m64 y) { x = reduce(u128(x) * y.x); return *this; }
23     m64 operator+(m64 y) const { return m64(*this) += y; }
24     m64 operator-(m64 y) const { return m64(*this) -= y; }
25     m64 operator*(m64 y) const { return m64(*this) *= y; }
26     bool operator==(m64 y) const { return (x >= m ? x - m : x) == (y.x >= m ? y.x - m : y.x); }
27     bool operator!=(m64 y) const { return not operator==(y); }
28     m64 pow(u64 n) const { m64 y = 1, z = *this; for (; n; n >>= 1, z *= z) if (n & 1) y *= z; return y; }
29 };
30
31 bool primetest(const uint64_t x) {
32     using u64 = uint64_t;
33     if (x == 2 or x == 3 or x == 5 or x == 7) return true;
34     if (x % 2 == 0 or x % 3 == 0 or x % 5 == 0 or x % 7 == 0) return false;

```

```

35     if (x < 121) return x > 1;
36     const u64 d = (x - 1) >> __builtin_ctzll(x - 1);
37     m64::set_mod(x);
38     const m64 one(1), minus_one(x - 1);
39     auto ok = [&](u64 a) {
40         auto y = m64(a).pow(d);
41         u64 t = d;
42         while (y != one and y != minus_one and t != x - 1) y *= y, t <<= 1;
43         if (y != minus_one and t % 2 == 0) return false;
44         return true;
45     };
46     if (x < (1ull << 32)) {
47         for (u64 a: {2, 7, 61}) if (not ok(a)) return false;
48     } else {
49         for (u64 a: {2, 325, 9375, 28178, 450775, 9780504, 1795265022}) {
50             if (x <= a) return true; if (not ok(a)) return false;
51         }
52     }
53     return true;
54 }
55 ll rho(ll n, ll c) {
56     m64::set_mod(n);
57     assert(n > 1);
58     const m64 cc(c);
59     auto f = [&](m64 x) { return x * x + cc; };
60     m64 x = 1, y = 2, z = 1, q = 1;
61     ll g = 1;
62     const ll m = 1LL << (std::__lg(n) / 5); // ?
63     for (ll r = 1; g == 1; r <<= 1) {
64         x = y;
65         for (ll _ = 0; _ < ll(r); ++_) y = f(y);
66         for (ll k = 0; k < r and g == 1; k += m) {
67             z = y;
68             for (ll _ = 0; _ < ll(std::min(m, r - k)); ++_) y = f(y), q *= x - y;
69             g = std::gcd(q.val(), n);
70         }
71     }
72     if (g == n) do {
73         z = f(z);
74         g = std::gcd((x - z).val(), n);
75     } while (g == 1);
76     return g;
77 }
78 std::mt19937_64 rng_mt{std::random_device{}}();
79 ll rnd(ll n) { return std::uniform_int_distribution<ll>(0, n - 1)(rng_mt); }
80 ll find_prime_factor(ll n) {
81     assert(n > 1);
82     if (primetest(n)) return n;
83     for (ll _ = 0; _ < 10011; ++_) {
84         ll m = rho(n, rnd(n));
85         if (primetest(m)) return m;
86         n = m;
87     }
88     std::cerr << "failed" << std::endl;
89     assert(false);
90     return -1;
91 }
92
93 //分解因数
94 vector<pair<ll, int>> factor(ll n) {
95     assert(n >= 1);
96     vector<pair<ll, int>> pf;

```

```
97     for (int p = 2; p < 100; ++p) {
98         if (p * p > n) break;
99         if (n % p == 0) {
100             int e = 0;
101             do { n /= p, e += 1; } while (n % p == 0);
102             pf.emplace_back(p, e);
103         }
104     }
105     while (n > 1) {
106         ll p = find_prime_factor(n);
107         ll e = 0;
108         do { n /= p, e += 1; } while (n % p == 0);
109         pf.emplace_back(p, e);
110     }
111     std::ranges::sort(pf);
112     return pf;
113 }
114 // 通过质因子分解因数
115 vector<pair<ll, int>> factor_by_lpf(ll n, vector<int>& lpf) {
116     vector<pair<ll, int>> res;
117     while (n > 1) {
118         int p = lpf[n];
119         int e = 0;
120         while (n % p == 0) {
121             n /= p;
122             ++e;
123         }
124         res.emplace_back(p, e);
125     }
126     return res;
127 }
```

radix_sort.hpp

```
1  template <const int N>
2  void radix_sort(int n, int a[]) {
3      static int b[N];
4      static int cnt[1 << 8];
5      memset(b, 0, sizeof b);
6      memset(cnt, 0, sizeof cnt);
7      static constexpr int mask = (1 << 8) - 1;
8      int *x = a, *y = b;
9      for (int i = 0; i < 32; i += 8) {
10         for (int j = 0; j != (1 << 8); ++j) cnt[j] = 0;
11         for (int j = 0; j != n; ++j) ++cnt[x[j] >> i & mask];
12         for (int sum = 0, j = 0; j != (1 << 8); ++j) {
13             sum += cnt[j], cnt[j] = sum - cnt[j];
14         }
15         for (int j = 0; j != n; ++j) y[cnt[x[j] >> i & mask]++] = x[j];
16         std::swap(x, y);
17     }
18 }
```

sieve.hpp

```
1  vector<int> minp, primes;
2  void sieve(int n) {
3      minp.assign(n + 1, 0);
4      primes.clear();
5      for (int i = 2; i < n + 1; i++) {
6          if (minp[i] == 0) {
```

```
7         minp[i] = i;
8         primes.emplace_back(i);
9     }
10    for (auto p : primes) {
11        if (i * p > n) {
12            break;
13        }
14        minp[i * p] = p;
15        if (p == minp[i]) {
16            break;
17        }
18    }
19 }
20 }
```

random

random.hpp

```

1  #include "../math/mod/modint.hpp"
2  namespace MeIoN_random_hash {
3      std::mt19937 RNG(std::chrono::steady_clock::now().time_since_epoch().count());
4      uint rng(uint limit) { return RNG() % limit; }
5      int rng(int l, int r) { return l + RNG() % (r - l); }
6      std::mt19937_64 RNG_64(std::chrono::steady_clock::now().time_since_epoch().count());
7      ull rng_64(ull limit) { return RNG_64() % limit; }
8      ll rng_64(ll l, ll r) { return l + RNG_64() % (r - l); }
9
10     using m1 = modint<998244353>;
11     using m2 = modint<1000000007>;
12
13     namespace get_prim {
14
15         constexpr ull md = (1ull << 61) - 1;
16
17         static inline constexpr ull modmul(const ull &a, const ull &b) {
18             u128 d = u128(a) * b;
19             ull ret = (ull(d) & md) + ull(d >> 61);
20             return ret >= md ? ret - md : ret;
21         }
22
23         static ull modpow(ull a, ull b) {
24             ull r = 1;
25             for (a %= md; b; a = modmul(a, a), b >>= 1) r = modmul(r, a);
26             return r;
27         }
28
29         static bool is_primitive(ull x) {
30             for (auto &d :
31                 vector<ull> {2, 3, 5, 7, 11, 13, 31, 41, 61, 151, 331, 1321})
32                 if (modpow(x, (md - 1) / d) <= 1) return false;
33             return true;
34         }
35
36         static ull get_basis() {
37             static auto rand_time =
38                 std::chrono::duration_cast<std::chrono::nanoseconds>(
39                     std::chrono::high_resolution_clock::now().time_since_epoch())
40                     .count();
41             static std::mt19937_64 rng(rand_time);
42             ull ret;
43             while (is_primitive(ret = rng() % (md - 1) + 1) == false);
44             return ret;
45         }
46     } using get_prim::get_basis;
47
48     template <typename T>
49     void shuffle(vector<T> &v) {
50         int n = v.size();
51         for (int i = 0; i < n; ++i) {
52             int j = rng(0, i + 1);
53             if (i != j) std::swap(v[i], v[j]);
54         }
55     }
56
57     void random_relabel(int n, vector<pair<int, int>> &v) {

```

```

58     shuffle(v);
59     vector<int> a(n);
60     std::iota(a.begin(), a.end(), 0);
61     shuffle(a);
62     for (auto &[x, y] : v) {
63         x = a[x], y = a[y];
64     }
65 }
66
67 template <int DIRECTED>
68 vector<pair<int, int>> random_graph(int n, bool simple) {
69     vector<pair<int, int>> v, cand;
70     for (int i = 0; i < n; ++i) {
71         for (int k = 0; k < n; ++k) {
72             if (simple and i == k) continue;
73             if (not DIRECTED and i > k) continue;
74             cand.emplace_back(i, k);
75         }
76     }
77     int m = rng(0, (int)cand.size() + 1);
78     set<int> se;
79     for (int i = 0; i < n; ++m) {
80         while (true) {
81             int i = rng(0, (int)cand.size());
82             if (simple and se.count(i)) continue;
83             se.emplace(i);
84             auto [a, b] = cand[i];
85             v.emplace_back(a, b);
86             break;
87         }
88     }
89     random_relabel(n, v);
90     return v;
91 }
92
93 template <typename T>
94 ull hash_pair(const pair<T, T> &X) {
95     static ll hash_base = RNG_64();
96     if (hash_base == 0) hash_base = RNG_64();
97     return hash_base * X.first + X.second;
98 }
99
100 template <typename T>
101 pair<uint, uint> hash_vector(const vector<T> &v) {
102     static vector<pair<m1, m2>> hash_base;
103     int n = v.size();
104     while (hash_base.size() < n + 1) {
105         hash_base.emplace_back(rng(m1::get_mod()), rng(m2::get_mod()));
106     }
107     m1 h1;
108     m2 h2;
109     for (int i = 0; i < n; ++i) {
110         h1 += hash_base[i].first * m1(v[i]);
111         h2 += hash_base[i].second * m2(v[i]);
112     }
113     h1 += hash_base[n].first, h2 += hash_base[n].second;
114     return pair(h1.val, h2.val);
115 }
116
117 template <typename T, int K>
118 pair<uint, uint> hash_array(const array<T, K> &v) {
119     static array<pair<m1, m2>, K> hash_base;

```

```
120     if (hash_base[0] == pair(m1(0), m2(0))) {
121         for (int i = 0; i < K; ++i) {
122             hash_base[i] = {rng(m1::get_mod()), rng(m2::get_mod())};
123         }
124     }
125     m1 h1;
126     m2 h2;
127     for (int i = 0; i < K; ++i) {
128         h1 += hash_base[i].first * m1(v[i]);
129         h2 += hash_base[i].second * m2(v[i]);
130     }
131     return pair(h1.val, h2.val);
132 }
133
134 // https://uoj.ac/problem/763
135 struct rooted_tree_hash {
136     vector<vector<int>> v;
137     int n;
138     vector<ull> hash;
139     vector<int> dis;
140
141     static vector<ull> &xs() {
142         static vector<ull> _xs;
143         return _xs;
144     }
145
146     rooted_tree_hash(const vector<vector<int>> &_v, int root = 0)
147         : v(_v), n(_v.size()) {
148         hash.resize(n);
149         dis.resize(n);
150         while ((int)xs().size() <= n) xs().emplace_back(get_basis());
151         dfs(root, -1);
152     }
153
154 private:
155     int dfs(int n, int fa) {
156         int dp = 0;
157         for (const int &i : v[n]) {
158             if (i == fa) continue;
159             chmax(dp, dfs(i, n) + 1);
160         }
161         ull x = xs()[dp], h = 1;
162         for (const int &i : v[n]) {
163             if (i == fa) continue;
164             h = get_prim::modmul(h, (x + hash[i]) % get_prim::md);
165         }
166         hash[n] = h;
167         return dis[n] = dp;
168     }
169 };
170 }
```

string

SA.hpp

```

1  struct MeIoN_SA {
2      std::vector<int> p, rank;
3      MeIoN_SA(const std::vector<int> &s) : p(s.size()), rank(s.size()) {
4          const int n = s.size();
5          int k = 0;
6          std::iota(p.begin(), p.end(), 0);
7          std::ranges::sort(p, {}, [&](int i) { return s[i]; });
8          for (int i = 0; i < n; ++i) {
9              rank[p[i]] = i and s[p[i]] == s[p[i - 1]] ? rank[p[i - 1]] : k++;
10         }
11         std::vector<int> q, count, new_rank(n);
12         for (int m = 1; m < n; m <= 1) {
13             q.resize(m);
14             std::iota(q.begin(), q.end(), n - m);
15             for (int i : p) {
16                 if (i >= m) {
17                     q.push_back(i - m);
18                 }
19             }
20             count.assign(k, 0);
21             for (int i : rank) {
22                 ++count[i];
23             }
24             std::partial_sum(count.begin(), count.end(), count.begin());
25             for (int i = n - 1; ~i; --i) {
26                 p[--count[rank[q[i]]]] = q[i];
27             }
28             auto cmp = [&](int i, int k) {
29                 int rk_i = i + m < n ? rank[i + m] : -1;
30                 int rk_k = k + m < n ? rank[k + m] : -1;
31                 return rank[i] == rank[k] and rk_i == rk_k;
32             };
33             k = 0;
34             for (int i = 0; i < n; ++i) {
35                 new_rank[p[i]] =
36                     i and cmp(p[i], p[i - 1]) ? new_rank[p[i - 1]] : k++;
37             }
38             rank.swap(new_rank);
39         }
40     }
41 };

```

SAM.hpp

```

1  namespace MeIoN_SAM {
2      static constexpr int ALPHABET = 26;
3      struct Node : std::array<int, ALPHABET> {
4          int link, len;
5          Node() : link(-1), len(0) { fill(-1); }
6      };
7      struct SAM : std::vector<Node> {
8          SAM() : std::vector<Node> (1) {};
9          SAM(const int n) : std::vector<Node> (1) { reserve(n); };
10         int ext(int p, int c) {
11             int pla = size();
12             emplace_back();
13             back().len = at(p).len + 1;

```



```

14     while (~p and at(p)[c] == -1) {
15         at(p)[c] = pla;
16         p = at(p).link;
17     }
18     if (~p) {
19         int q = at(p)[c];
20         if (at(p).len + 1 == at(q).len) {
21             back().link = q;
22         } else {
23             int cp = size();
24             push_back(at(q));
25             back().len = at(p).len + 1;
26             while (~p and at(p)[c] == q) {
27                 at(p)[c] = cp;
28                 p = at(p).link;
29             }
30             at(q).link = at(pla).link = cp;
31         }
32     } else {
33         back().link = 0;
34     }
35     return pla;
36 }
37 std::tuple<vector<int>, vector<vector<int>>> build(const string &s) {
38     const int n = s.length();
39     vector<int> sz(n << 1);
40     for (int pla = 0; const char c : s) {
41         pla = ext(pla, c - 'a');
42         sz[pla] = 1;
43     }
44     vector<vector<int>> v(n << 1);
45     for (int i = 1; i < size(); ++i) {
46         v[at(i).link].emplace_back(i);
47     }
48     auto dfs = [&](auto &&se, int n) -> void {
49         for (int i : v[n]) {
50             se(se, i);
51             sz[n] += sz[i];
52         }
53     };
54     dfs(dfs, 0);
55     return {sz, v};
56 }
57 };
58 } using MeIoN_SAM::SAM;

```

acam.hpp

```

1 struct MeIoN_ACAM {
2     static constexpr int ALPHABET = 26;
3     struct Node {
4         int len, fail;
5         std::array<int, ALPHABET> next;
6         Node() : len { 0 }, fail { 0 }, next {} {}
7     };
8     std::vector<Node> t;
9     MeIoN_ACAM() {
10         init();
11     }
12     void init() {
13         t.assign(2, Node());
14         t[0].next.fill(1);

```

```

15     t[0].len = -1;
16 }
17 int newNode() {
18     t.emplace_back();
19     return t.size() - 1;
20 }
21 int add(const std::string& a) {
22     int p = 1;
23     for (auto c : a) {
24         int x = c - 'a';
25         if (t[p].next[x] == 0) {
26             t[p].next[x] = newNode();
27             t[t[p].next[x]].len = t[p].len + 1;
28         }
29         p = t[p].next[x];
30     }
31     return p;
32 }
33 void work() {
34     std::queue<int> q;
35     q.push(1);
36     while (!q.empty()) {
37         int x = q.front();
38         q.pop();
39
40         for (int i = 0; i < ALPHABET; i++) {
41             if (t[x].next[i] == 0) {
42                 t[x].next[i] = t[t[x].fail].next[i];
43             } else {
44                 t[t[x].next[i]].fail = t[t[x].fail].next[i];
45                 q.push(t[x].next[i]);
46             }
47         }
48     }
49 }
50 int next(int p, int x) { return t[p].next[x]; }
51 int fail(int p)         { return t[p].fail; }
52 int len(int p)          { return t[p].len; }
53 int size()              { return t.size(); }
54 };
55 using AC = MeIoN_ACAM;

```

hash.hpp

```

1 namespace getmod {
2     bool guidingstar_ckpr(int n) {
3         if (n < 1) return false;
4         for (int i = 2, ed = n; i * i <= ed; ++i) {
5             if (n % i == 0) return false;
6         }
7         return true;
8     }
9     int guidingstar_find_pr(int n) {
10         while (not guidingstar_ckpr(n)) ++n;
11         return n;
12     }
13     const int m1 = guidingstar_find_pr(rng() % 900000000 + 100000000),
14             m2 = guidingstar_find_pr(rng() % 900000000 + 100000000);
15     constexpr int M1 = 1000000123, M2 = 1000000181;
16 }
17 struct rolling_HASH {
18     int n;

```

```

19     vector<pair<int, int>> h, p;
20     rolling_HASH(const string &s = "") : n(s.length()), h(n + 1), p(n + 1) {
21         for (int i = 0; i < n; ++i) {
22             h[i + 1].first = (13111 * h[i].first + s[i] - '0') % getmod::m1;
23             h[i + 1].second = (13111 * h[i].second + s[i] - '0') % getmod::m2;
24         }
25         p[0] = {1, 1};
26         for (int i = 0; i < n; ++i) {
27             p[i + 1].first = 13111 * p[i].first % getmod::m1;
28             p[i + 1].second = 13111 * p[i].second % getmod::m2;
29         }
30     }
31     pair<ll, ll> get(int l, int r) const {
32         return {
33             (h[r].first + 111 * (getmod::m1 - h[l].first) * p[r - l].first) %
34             getmod::m1,
35             (h[r].second + 111 * (getmod::m2 - h[l].second) * p[r - l].second) %
36             getmod::m2};
37     }
38 };
39 struct HASH {
40     int n;
41     vector<pair<int, int>> h, p;
42     HASH(const string &s = "") : n(s.length()), h(n + 1), p(n + 1) {
43         for (int i = 0; i < n; ++i) {
44             h[i + 1].first = (13111 * h[i].first + s[i] - '0') % getmod::M1;
45             h[i + 1].second = (13111 * h[i].second + s[i] - '0') % getmod::M2;
46         }
47         p[0] = {1, 1};
48         for (int i = 0; i < n; ++i) {
49             p[i + 1].first = 13111 * p[i].first % getmod::M1;
50             p[i + 1].second = 13111 * p[i].second % getmod::M2;
51         }
52     }
53     pair<ll, ll> get(int l, int r) const {
54         return {
55             (h[r].first + 111 * (getmod::M1 - h[l].first) * p[r - l].first) %
56             getmod::M1,
57             (h[r].second + 111 * (getmod::M2 - h[l].second) * p[r - l].second) %
58             getmod::M2};
59     }
60 };
61 template<typename HASH>
62 int get_lcp(const HASH &h1, int l1, int r1, const HASH &h2, int l2, int r2) {
63     int sz = std::min(r1 - l1, r2 - l2);
64     int l = 0, r = sz + 1;
65     while (r - l > 1) {
66         int m = l + r >> 1;
67         if (h1.get(l1, l1 + m) == h2.get(l2, l2 + m)) {
68             l = m;
69         } else {
70             r = m;
71         }
72     }
73     return l;
74 };
75 template <typename HASH>
76 bool hash_same(const HASH &h1, int l1, const HASH &h2, int l2, int sz) {
77     return (l1 + sz <= h1.n and l2 + sz <= h2.n) and
78            h1.get(l1, l1 + sz) == h2.get(l2, l2 + sz);
79 }

```

manache.hpp

```
1 namespace MeIoN_namache{
2     ll n;
3     constexpr int working_sz = 1145141;
4     int p[working_sz];
5     string ss;
6     void namache(string &s) {
7         s = " " + s;
8         ss = "";
9         ss += '&';
10        for (int i = 1; i <= n; i++) {
11            ss += '#';
12            ss += s[i];
13        }
14        ss += '#';
15        ss += '*';
16        for (int i = 1; i < ss.size(); i++) p[i] = 1;
17        for (int i = 1, l = 1, r = 0; i + 1 < ss.size(); i++) {
18            if (i <= r) p[i] = std::min(r - i + 1, p[l + r - i]);
19            while (ss[i - p[i]] == ss[i + p[i]]) p[i] ++;
20            if (i + p[i] - 1 > r) {
21                l = i - p[i] + 1;
22                r = i + p[i] - 1;
23            }
24        }
25    }
26    // [l, r]
27    bool check(int l, int r) {
28        int len = r - l + 1;
29        l *= 2, r *= 2;
30        return p[l + r >> 1] - 1 >= len;
31    }
32 }using namespace MeIoN_namache;
```

trie

```
1 template <int W>
2 struct trie {
3     struct node {
4         array<int, W> ch,
5             nxt;
6         int fa;
7         int link;
8         node() : fa(-1), link(-1) {
9             ch.fill(-1);
10            nxt.fill(-1);
11        }
12    };
13    int n_node;
14    vector<node> nodes;
15    vector<int> words;
16    vector<int> bfs;
17
18    trie() :n_node(0) {
19        new_node();
20    }
21
22    node &operator[](int i) { return nodes[i]; }
23
24    template <typename container>
```

```
25     int add(container s, int off) {
26         int pla = 0;
27         for (auto &&c : s) {
28             pla = add_single(pla, c, off);
29         }
30         words.emplace_back(pla);
31         return pla;
32     }
33
34     int add_single(int pla, int c, int off) {
35         c -= off;
36         assert(-1 < c and c < W);
37         if (nodes[pla].ch[c] != -1) return nodes[pla].ch[c];
38         nodes[pla].ch[c] = new_node();
39         nodes.back().fa = pla;
40         return nodes[pla].ch[c];
41     }
42
43     void calc_suffix_link() {
44         bfs.resize(n_node);
45         int p = 0, q = 0;
46         bfs[q++] = 0;
47         nodes[0].nxt.fill(0);
48         while (p < q) {
49             int v = bfs[p++];
50             if (v) nodes[v].nxt = nodes[nodes[v].link].nxt;
51             for (int i = 0; i < W; ++i) {
52                 int w = nodes[v].ch[i];
53                 if (w == -1) continue;
54                 nodes[w].link = nodes[v].nxt[i];
55                 nodes[v].nxt[i] = w;
56                 bfs[q++] = w;
57             }
58         }
59     }
60
61     vector<int> calc_count() {
62         vector<int> count(n_node);
63         for (int i : words) {
64             ++count[i];
65         }
66         for (int i : bfs) {
67             if (i) {
68                 count[i] += count[nodes[i].link];
69             }
70         }
71         return count;
72     }
73
74 private:
75     int new_node() {
76         node c;
77         nodes.emplace_back(c);
78         return n_node++;
79     }
80 };
```

tree

LCA.hpp

```

1  template <const int N> struct LCA {
2  public:
3      LCA (const vector<vector<int>>> &v, int rt) :
4          sz(v.size()), root(rt), up(sz), dis(sz), lg(0) {
5          for (auto &i : up) i.fill(0);
6          while ((1 << lg) <= sz) lg++;
7          assert(lg <= N);
8          auto dfs = [&](auto &&se, int n, int fa, int dp) -> void {
9              dis[n] = dp;
10             up[n][0] = fa;
11             for (int i = 1; i <= lg - 1; i++) up[n][i] = up[up[n][i - 1]][i - 1];
12             for (const auto &x : v[n]) {
13                 if (x == fa) continue;
14                 se(se, x, n, dp + 1);
15             }
16         };
17         dfs(dfs, rt, rt, 0);
18     }
19     int &operator[](const int &x) { return up[x]; }
20     int jump(int x, int tp) {
21         chmin(tp, dis[x] + 1);
22         for (int i = 0; i < lg; i++) {
23             if (tp >> i & 1) {
24                 x = up[x][i];
25             }
26         }
27         return up[x][0];
28     }
29     int lca(int x, int y) {
30         if (dis[x] < dis[y])
31             std::swap(x, y);
32         int z = dis[x] - dis[y];
33         for (int i = 0; i < lg; i++) {
34             if (z >> i & 1) {
35                 x = up[x][i];
36             }
37         }
38         if (x == y) return x;
39         for (int i = lg; i--;) {
40             int X = up[x][i], Y = up[y][i];
41             if (X != Y) x = X, y = Y;
42         }
43         return up[x][0];
44     }
45     int dist(int x, int y) {
46         return dis[x] + dis[y] - 2 * dis[lca(x, y)];
47     }
48 private:
49     int root, sz, lg;
50     std::vector<std::array<int, N>> up;
51     std::vector<int> dis;
52 };

```

LTT.hpp

```

1  vector<int> get_fa(const vector<vector<int>>> &v, int s) {
2      int n = v.size();

```

```

3     vector<int> pos(n, -1), p, label(n), dom(n), sdom(n), dsu(n), par(n);
4     vector<vector<int>>> rg(n), bucket(n);
5     auto dfs = [&] (auto &&se, int n)->void {
6         int t = p.size();
7         p.emplace_back(n);
8         label[t] = sdom[t] = dsu[t] = pos[n] = t;
9         for (const int i : v[n]) {
10             if (pos[i] == -1) {
11                 se(se, i);
12                 par[pos[i]] = t;
13             }
14             rg[pos[i]].emplace_back(t);
15         }
16     };
17     auto find = [&] (auto &&se, int n, int x) {
18         if (n == dsu[n]) return x ? -1 : n;
19         int v = se(se, dsu[n], x + 1);
20         if (v < 0) return n;
21         if (sdom[label[dsu[n]]] < sdom[label[n]]) {
22             label[n] = label[dsu[n]];
23         }
24         dsu[n] = v;
25         return x ? v : label[n];
26     };
27     dfs(dfs, s);
28     std::iota(dom.begin(), dom.end(), 0);
29     for (int i = (int)p.size() - 1; ~i; --i) {
30         for (int k : rg[i]) {
31             chmin(sdom[i], sdom[find(find, k, 0)]);
32         }
33         if (i) {
34             bucket[sdom[i]].emplace_back(i);
35         }
36         for (int k : bucket[i]) {
37             int j = find(find, k, 0);
38             dom[k] = sdom[j] == sdom[k] ? sdom[j] : j;
39         }
40         if (i > 1) {
41             dsu[i] = par[i];
42         }
43     }
44     for (int i = 1; i < (int)p.size(); ++i) {
45         if (dom[i] != sdom[i]) {
46             dom[i] = dom[dom[i]];
47         }
48     }
49     vector<int> res(n, -1);
50     res[s] = s;
51     for (int i = 1; i < (int)p.size(); ++i) {
52         res[p[i]] = p[dom[i]];
53     }
54     return res;
55 }

```

centroid.hpp

```

1     vector<int> centroid(const vector<vector<int>>> &v) {
2         const int n = (int)v.size();
3         vector<pair<int, int>>> st;
4         vector<int> sz(n), ff(n);
5
6         st.reserve(n);

```

```
7     st.emplace_back(0, -1);
8     while (not st.empty()) {
9         const auto [n, fa] = st.back();
10        if (sz[n] == 0) {
11            sz[n] = 1;
12            for (const int i : v[n]) {
13                if (i == fa) continue;
14                st.emplace_back(i, n);
15            }
16        } else {
17            for (const int i : v[n]) {
18                if (i == fa) continue;
19                sz[n] += sz[i];
20            }
21            ff[n] = fa;
22            st.pop_back();
23        }
24    }
25
26    vector<int> ret;
27    ret.reserve(8);
28    int size = n;
29    for (int i = 0; i < n; ++i) {
30        int val = n - sz[i];
31        for (const int x : v[i]) {
32            if (x == ff[i]) continue;
33            chmax(val, sz[i]);
34        }
35        if (chmin(size, val)) {
36            ret.clear();
37        }
38        if (val == size) {
39            ret.emplace_back(i);
40        }
41    }
42    return ret;
43 }
```

unrooted_tree_hash.hpp

```
1
2 #include "centroid.hpp"
3 #include "../random/random.hpp"
4
5 using MeIoN_random_hash::rooted_tree_hash;
6
7 vector<ull> unrooted_tree_hash(const vector<vector<int>> &v) {
8     vector root = centroid(v);
9     if (root.size() == 1) root.emplace_back(root[0]);
10    vector<ull> res;
11    for (const int x : root) {
12        res.emplace_back(rooted_tree_hash(v, x).hash[x]);
13    }
14    sort(res);
15    return res;
16 }
```

最小斯坦纳树

```
1 NAME MeIoN_is_UMP45() {
2     std::cin >> n >> m >> k;
```



```
3     vector<vector<pair<int, int>>> v(n + 1);
4     for (int i = 0, l, r, w; i < m; ++i) {
5         std::cin >> l >> r >> w;
6         v[l].emplace_back(r, w);
7         v[r].emplace_back(l, w);
8     }
9     A<A<int, 101>, 1 << 11> dp;
10    for (auto &v : dp) v.fill(__INT_MAX__);
11    vector<int> S(k);
12    for (int c = 0; auto &i : S) {
13        std::cin >> i;
14        dp[1 << c++][i] = 0;
15    }
16    rpq<pair<int, int>> q;
17    auto dij = [&](int BE) {
18        while (not q.empty()) {
19            int n = q.top().second;
20            q.pop();
21            for (const auto & [ i, w ] : v[n]) {
22                if (dp[BE][i] > dp[BE][n] + w) {
23                    dp[BE][i] = dp[BE][n] + w;
24                    q.emplace(dp[BE][i], i);
25                }
26            }
27            while (not q.empty() and q.top().first != dp[BE][q.top().second])
28                q.pop();
29        }
30    };
31    for (int i = 1; i < 1 << k; ++i) {
32        for (int k = 1; k <= n; ++k) {
33            for (int j = i & (i - 1); j; j = i & (j - 1)) {
34                dp[i][k] = std::min(dp[i][k], dp[j][k] + dp[i ^ j][k]);
35            }
36            if (dp[i][k] < __INT_MAX__) q.emplace(dp[i][k], k);
37        }
38        dij(i);
39    }
40    int ans = __INT_MAX__;
41    for (int i = 1; i <= n; ++i) {
42        ans = std::min(ans, dp[(1 << k) - 1][i]);
43    }
44    std::cout << ans << '\n';
45 }
```

a_monoid

max_add.hpp

```
1  #include "../monoid/add.hpp"
2  #include "../monoid/max.hpp"
3
4  template <typename E>
5  struct a_monoid_max_add {
6      using Monoid_X = monoid_max<E>;
7      using Monoid_A = monoid_add<E>;
8      using X = typename Monoid_X::value_type;
9      using A = typename Monoid_A::value_type;
10     static constexpr X act(const X &x, const A &a, const ll &size) {
11         if (x == inf<E>) return x;
12         return x + a;
13     }
14 };
```

min_add.hpp

```
1  #include "../monoid/add.hpp"
2  #include "../monoid/min.hpp"
3
4  template <typename E>
5  struct a_monoid_min_add {
6      using Monoid_X = monoid_min<E>;
7      using Monoid_A = monoid_add<E>;
8      using X = typename Monoid_X::value_type;
9      using A = typename Monoid_A::value_type;
10     static constexpr X act(const X &x, const A &a, const ll &size) {
11         if (x == inf<E>) return x;
12         return x + a;
13     }
14 };
```

minidx_add.hpp

```
1  #include "../monoid/add.hpp"
2  #include "../monoid/min_idx.hpp"
3
4  template <typename E, bool tie_is_left = true>
5  struct a_monoid_min_idx_add {
6      using Monoid_X = monoid_min_idx<E, tie_is_left>;
7      using Monoid_A = monoid_add<E>;
8      using X = typename Monoid_X::value_type;
9      using A = typename Monoid_A::value_type;
10     static constexpr X act(const X &x, const A &a, const ll &size) {
11         if (x.first == inf<E>) return x;
12         return {x.first + a, x.second};
13     }
14 };
```

sum_add.hpp

```
1  #include "../monoid/add.hpp"
2
3  template <typename E>
4  struct a_monoid_sum_add {
5      using Monoid_X = monoid_add<E>;
6      using Monoid_A = monoid_add<E>;
```

```
7     using X = typename Monoid_X::value_type;
8     using A = typename Monoid_A::value_type;
9     static constexpr X act(const X &x, const A &a, const ll &size) {
10         return x + a * E(size);
11     }
12 };
```

monoid

add.hpp

```

1
2  template <typename E>
3  struct monoid_add {
4      using X = E;
5      using value_type = X;
6      static constexpr X op(const X &x, const X &y) noexcept { return x + y; }
7      static constexpr X inverse(const X &x) noexcept { return -x; }
8      static constexpr X power(const X &x, ll n) noexcept { return X(n) * x; }
9      static constexpr X unit() { return X(0); }
10     static constexpr bool commute = true;
11 };

```

add_array.hpp

```

1
2  template <typename E, int K>
3  struct monoid_add_array {
4      using value_type = array<E, K>;
5      using X = value_type;
6      static X op(X x, X y) {
7          for (int i = 0; i < K; ++i) x[i] += y[i];
8          return x;
9      }
10     static constexpr X unit() { return X {}; }
11     static constexpr X inverse(X x) {
12         for (auto& v : x) v = -v;
13         return x;
14     }
15     static constexpr X power(X x, ll n) {
16         for (auto& v : x) v *= E(n);
17         return x;
18     }
19     static constexpr bool commute = 1;
20 };

```

add_pair.hpp

```

1
2  template <typename E>
3  struct monoid_add_pair {
4      using value_type = pair<E, E>;
5      using X = value_type;
6      static constexpr X op(const X &x, const X &y) {
7          return {x.fi + y.fi, x.se + y.se};
8      }
9      static constexpr X inverse(const X &x) { return {-x.fi, -x.se}; }
10     static constexpr X unit() { return {0, 0}; }
11     static constexpr bool commute = true;
12 };

```

gcd.hpp

```

1
2  template <class X>
3  struct monoid_gcd {
4      using value_type = X;
5      static constexpr X op(const X &a, const X &b) noexcept { return std::gcd(a, b); }

```

```
6     static constexpr X unit() { return 0; }
7     static constexpr bool commute = true;
8 };
```

max.hpp

```
1
2 template <class X>
3 struct monoid_max {
4     using value_type = X;
5     static constexpr X op(const X &a, const X &b) noexcept { return std::max(a, b); }
6     static constexpr X unit() { return -std::numeric_limits<X>::max(); }
7     static constexpr bool commute = true;
8 };
```

max_idx.hpp

```
1
2 template <typename T, bool tie_is_left = true>
3 struct monoid_max_idx {
4     using value_type = pair<T, int>;
5     using X = value_type;
6     static X op(X x, X y) {
7         if (x.first > y.first) return x;
8         if (x.first < y.first) return y;
9         if (x.second > y.second) std::swap(x, y);
10        return (tie_is_left ? x : y);
11    }
12    static constexpr X unit() { return {-INTMAX, -1}; }
13    static constexpr bool commute = true;
14 };
```

min.hpp

```
1
2 template <class X>
3 struct monoid_min {
4     using value_type = X;
5     static constexpr X op(const X &a, const X &b) noexcept { return std::min(a, b); }
6     static constexpr X unit() { return std::numeric_limits<X>::max(); }
7     static constexpr bool commute = true;
8 };
```

min_idx.hpp

```
1
2 template <typename T, bool tie_is_left = true>
3 struct monoid_min_idx {
4     using value_type = pair<T, int>;
5     using X = value_type;
6     static constexpr bool is_small(const X &x, const X &y) {
7         if (x.first < y.first) return true;
8         if (x.first > y.first) return false;
9         return (tie_is_left ? (x.second < y.second) : (x.second >= y.second));
10    }
11    static X op(X x, X y) { return (is_small(x, y) ? x : y); }
12    static constexpr X unit() { return {INTMAX, -1}; }
13    static constexpr bool commute = true;
14 };
```

sum.hpp

```
1
2  template <class X>
3  struct monoid_sum {
4      using value_type = X;
5      static constexpr X op(const X & a, const X &b) noexcept { return a + b; }
6      static constexpr X unit() { return 0; }
7      static constexpr bool commute = true;
8  };
```

xor.hpp

```
1
2  template <typename X>
3  struct monoid_xor {
4      using value_type = X;
5      static X op(X x, X y) { return x ^ y; };
6      static constexpr X inverse(const X &x) noexcept { return x; }
7      static constexpr X power(const X &x, ll n) noexcept {
8          return (n & 1 ? x : 0);
9      }
10     static constexpr X unit() { return X(0); };
11     static constexpr bool commute = true;
12 };
```

seg

lazy_seg_base.hpp

```

1  template <typename a_monoid>
2  struct lazy_seg {
3      using AM = a_monoid;
4      using MX = typename AM::Monoid_X;
5      using MA = typename AM::Monoid_A;
6      using X = typename MX::value_type;
7      using A = typename MA::value_type;
8      int n, log, size;
9      vector<X> dat;
10     vector<A> tag;
11
12     lazy_seg() {}
13     lazy_seg(int n) { build(n); }
14     template <typename F>
15     lazy_seg(int n, F f) {
16         build(n, f);
17     }
18     lazy_seg(const vector<X> &v) { build(v); }
19
20     void build(int m) {
21         build(m, [](int i) -> X { return MX::unit(); });
22     }
23     void build(const vector<X> &v) {
24         build(v.size(), [&](int i) -> X { return v[i]; });
25     }
26     template <typename F>
27     void build(int m, F f) {
28         n = m, log = 1;
29         while ((1 << log) < n) ++log;
30         size = 1 << log;
31         dat.assign(size << 1, MX::unit());
32         tag.assign(size, MA::unit());
33         for (int i = 0; i < n; ++i) dat[size + i] = f(i);
34         for (int i = size - 1; i > 0; --i) update(i);
35     }
36
37     void update(int k) { dat[k] = MX::op(dat[2 * k], dat[2 * k + 1]); }
38     void set(int p, X x) {
39         assert(-1 < p and p < n);
40         p += size;
41         for (int i = log; i > 0; --i) push(p >> i);
42         dat[p] = x;
43         for (int i = 1; i < log + 1; ++i) update(p >> i);
44     }
45     void multiply(int p, const X &x) {
46         assert(-1 < p and p < n);
47         p += size;
48         for (int i = log; i > 0; --i) push(p >> i);
49         dat[p] = MX::op(dat[p], x);
50         for (int i = 1; i < log + 1; ++i) update(p >> i);
51     }
52
53     X get(int p) {
54         assert(p > -1 and p < n);
55         p += size;
56         for (int i = log; i > 0; --i) push(p >> i);
57         return dat[p];

```

```

58     }
59
60     vector<X> get_all() {
61         for (int i = 1; i < size; ++i) push(i);
62         return {dat.begin() + size, dat.begin() + size + n};
63     }
64
65     X prod(int l, int r) {
66         assert(-1 < l and l < r + 1 and r < n + 1);
67         if (l == r) return MX::unit();
68         l += size, r += size;
69         for (int i = log; i > 0; --i) {
70             if (((l >> i) << i) != l) push(l >> i);
71             if (((r >> i) << i) != r) push((r - 1) >> i);
72         }
73         X x1 = MX::unit(), xr = MX::unit();
74         while (l < r) {
75             if (l & 1) x1 = MX::op(x1, dat[l++]);
76             if (r & 1) xr = MX::op(dat[--r], xr);
77             l >>= 1, r >>= 1;
78         }
79         return MX::op(x1, xr);
80     }
81
82     X prod_all() { return dat[1]; }
83
84     void apply(int l, int r, A a) {
85         assert(-1 < l and l < r + 1 and r < n + 1);
86         if (l == r) return;
87         l += size, r += size;
88         for (int i = log; i >= 1; i--) {
89             if (((l >> i) << i) != l) push(l >> i);
90             if (((r >> i) << i) != r) push((r - 1) >> i);
91         }
92         int l2 = l, r2 = r;
93         while (l < r) {
94             if (l & 1) apply_at(l++, a);
95             if (r & 1) apply_at(--r, a);
96             l >>= 1, r >>= 1;
97         }
98         l = l2, r = r2;
99         for (int i = 1; i <= log; i++) {
100             if (((l >> i) << i) != l) update(l >> i);
101             if (((r >> i) << i) != r) update((r - 1) >> i);
102         }
103     }
104
105     template <typename F>
106     int max_right(const F check, int l) {
107         assert(0 <= l && l <= n);
108         assert(check(MX::unit()));
109         if (l == n) return n;
110         l += size;
111         for (int i = log; i >= 1; i--) push(l >> i);
112         X sm = MX::unit();
113         do {
114             while (l % 2 == 0) l >>= 1;
115             if (not check(MX::op(sm, dat[l]))) {
116                 while (l < size) {
117                     push(l);
118                     l = (2 * l);
119                     if (check(MX::op(sm, dat[l]))) {

```



```

120         sm = MX::op(sm, dat[l++]);
121     }
122 }
123     return l - size;
124 }
125     sm = MX::op(sm, dat[l++]);
126 } while ((l & -l) != 1);
127     return n;
128 }
129
130 template <typename F>
131 int min_left(const F check, int r) {
132     assert(0 <= r && r <= n);
133     assert(check(MX::unit()));
134     if (r == 0) return 0;
135     r += size;
136     for (int i = log; i >= 1; i--) push((r - 1) >> i);
137     X sm = MX::unit();
138     do {
139         r--;
140         while (r > 1 && (r % 2)) r >>= 1;
141         if (!check(MX::op(dat[r], sm))) {
142             while (r < size) {
143                 push(r);
144                 r = (2 * r + 1);
145                 if (check(MX::op(dat[r], sm))) {
146                     sm = MX::op(dat[r--], sm);
147                 }
148             }
149             return r + 1 - size;
150         }
151         sm = MX::op(dat[r], sm);
152     } while ((r & -r) != r);
153     return 0;
154 }
155
156 private:
157 void apply_at(int k, A a) {
158     int sz = 1 << (log - topbit(k));
159     dat[k] = AM::act(dat[k], a, sz);
160     if (k < size) tag[k] = MA::op(tag[k], a);
161 }
162 void push(int k) {
163     if (tag[k] == MA::unit()) return;
164     apply_at(2 * k, tag[k]), apply_at(2 * k + 1, tag[k]);
165     tag[k] = MA::unit();
166 }
167 };

```

seg_base.hpp

```

1  template <class monoid>
2  struct Seg {
3      using MX = monoid;
4      using X = typename MX::value_type;
5      using value_type = X;
6      vector<X> dat;
7      int n, log, sz;
8      Seg() {}
9      Seg(int n) { build(n); }
10     template <typename F>
11     Seg(int n, F f) { build(n, f); }

```

```

12 Seg(const vector<X> &v) { build(v); }
13 void build(int m) { build(m, [](int i) ->X { return MX::unit(); }); }
14 void build(const vector<X> &v) { build(int(v.size()), [&](int i) -> X { return v[i]; }); }
15 template <typename F>
16 void build(int N, F f) {
17     n = N;
18     while ((1 << log) < n) ++log;
19     sz = 1 << log;
20     dat.assign(sz << 1, MX::unit());
21     for (int i = 0; i < n; ++i) dat[sz + i] = f(i);
22     for (int i = sz - 1; i >= 1; --i) update(i);
23 }
24 X get(int i) { return dat[sz + i]; }
25 vector<X> get_all() { return {dat.begin() + sz, dat.begin() + sz + n}; }
26 void update(int i) { dat[i] = monoid::op(dat[2 * i], dat[2 * i + 1]); }
27 void set(int i, const X &x) {
28     dat[i += sz] = x;
29     while (i >= 1) update(i);
30 }
31 void multiply(int i, const X &x) {
32     i += sz;
33     dat[i] = monoid::op(dat[i], x);
34     while (i >= 1) update(i);
35 }
36 X prod(int l, int r) {
37     X vl = monoid::unit(), vr = monoid::unit();
38     l += sz, r += sz;
39     while (l < r) {
40         if (l & 1) vl = monoid::op(vl, dat[l++]);
41         if (r & 1) vr = monoid::op(dat[--r], vr);
42         l >>= 1, r >>= 1;
43     }
44     return monoid::op(vl, vr);
45 }
46 X prod_all() { return dat[1]; }
47 template <class F>
48 int max_right(F check, int l) {
49     if (l == n) return n;
50     l += sz;
51     X sm = monoid::unit();
52     do {
53         while (l % 2 == 0) l >>= 1;
54         if (not check(monoid::op(sm, dat[l]))) {
55             while (l < sz) {
56                 l = 2 * l;
57                 if (check(monoid::op(sm, dat[l]))) { sm = monoid::op(sm, dat[l++]); }
58             }
59             return l - sz;
60         }
61         sm = monoid::op(sm, dat[l++]);
62     } while ((l & -l) != 1);
63     return n;
64 }
65 template <class F>
66 int min_left(F check, int r) {
67     if (r == 0) return 0;
68     r += sz;
69     X sm = monoid::unit();
70     do {
71         --r;
72         while (r > 1 and (r % 2)) r >>= 1;
73         if (not check(monoid::op(dat[r], sm))) {

```

```
74         while (r < sz) {
75             r = 2 * r + 1;
76             if (check(monoid::op(dat[r], sm))) { sm = monoid::op(dat[r--], sm); }
77         }
78         return r + 1 - sz;
79     }
80     sm = monoid::op(dat[r], sm);
81 } while ((r & -r) != r);
82 return 0;
83 }
84 X xor_prod(int l, int r, int xor_val) {
85     static_assert(monoid::commute);
86     X x = monoid::unit();
87     for (int k = 0; k < log + 1; ++k) {
88         if (l >= r) break;
89         if (l & 1) { x = monoid::op(x, dat[(sz >> k) + ((l++) ^ xor_val)]); }
90         if (r & 1) { x = monoid::op(x, dat[(sz >> k) + ((--r) ^ xor_val)]); }
91         l /= 2, r /= r, xor_val /= 2;
92     }
93     return x;
94 }
95 };
```

mod

modint.hpp

```

1  struct has_mod_impl {
2      template <class T>
3          static auto check(T&& x) -> decltype(x.get_mod(), std::true_type {});
4      template <class T>
5          static auto check(...) -> std::false_type;
6  };
7  template <class T>
8  class has_mod : public decltype(has_mod_impl::check<T>(std::declval<T>())) { };
9  template <int mod>
10 struct modint {
11     static constexpr bool is_mod_int = true;
12     static constexpr unsigned umod = unsigned(mod);
13     static_assert(umod < unsigned(1) << 31);
14     int val;
15     static modint raw(unsigned v) {
16         modint x;
17         x.val = v;
18         return x;
19     }
20     constexpr modint(const ll val = 0) noexcept : val(val >= 0 ? val % mod : (mod - (-val) % mod) % mod) { }
21     bool operator<(const modint& other) const { return val < other.val; }
22     modint& operator+=(const modint& p) {
23         if ((val += p.val) >= mod)
24             val -= mod;
25         return* this;
26     }
27     modint& operator-=(const modint& p) {
28         if ((val += mod - p.val) >= mod)
29             val -= mod;
30         return* this;
31     }
32     modint& operator*=(const modint& p) {
33         val = (int)(1LL * val * p.val % mod);
34         return* this;
35     }
36     modint& operator/=(const modint& p) {
37         *this *= p.inv();
38         return* this;
39     }
40     modint operator-() const { return modint::raw(val ? mod - val : unsigned(0)); }
41     modint operator+(const modint& p) const { return modint(*this) += p; }
42     modint operator-(const modint& p) const { return modint(*this) -= p; }
43     modint operator*(const modint& p) const { return modint(*this) *= p; }
44     modint operator/(const modint& p) const { return modint(*this) /= p; }
45     bool operator==(const modint& p) const { return val == p.val; }
46     bool operator!=(const modint& p) const { return val != p.val; }
47     friend std::istream& operator>>(std::istream& is, modint& p) {
48         ll x;
49         is >> x;
50         p = x;
51         return is;
52     }
53     friend std::ostream& operator<<(std::ostream& os, modint p) { return os << p.val; }
54     modint inv() const {
55         int a = val, b = mod, u = 1, v = 0, t;
56         while (b > 0)
57             t = a / b, std::swap(a -= t * b, b), std::swap(u -= t * v, v);

```

```

58     return modint(u);
59 }
60 modint ksm(ll n) const {
61     modint ret(1), mul(val);
62     while (n > 0) {
63         if (n & 1)
64             ret *= mul;
65         mul *= mul;
66         n >>= 1;
67     }
68     return ret;
69 }
70 static constexpr int get_mod() { return mod; }
71 static constexpr pair<int, int> ntt_info() {
72     if (mod == 120586241) return {20, 74066978};
73     if (mod == 167772161) return {25, 17};
74     if (mod == 469762049) return {26, 30};
75     if (mod == 754974721) return {24, 362};
76     if (mod == 880803841) return {23, 211};
77     if (mod == 943718401) return {22, 663003469};
78     if (mod == 998244353) return {23, 31};
79     if (mod == 1004535809) return {21, 836905998};
80     if (mod == 1045430273) return {20, 363};
81     if (mod == 1051721729) return {20, 330};
82     if (mod == 1053818881) return {20, 2789};
83     return { -1, -1 };
84 }
85 static constexpr bool can_ntt() { return ntt_info().first != -1; }
86 };
87 ll mod_inv(ll val, ll mod) {
88     if (mod == 0) return 0;
89     mod = std::abs(mod);
90     val %= mod;
91     if (val < 0) val += mod;
92     ll a = val, b = mod, u = 1, v = 0, t;
93     while (b > 0) {
94         t = a / b;
95         std::swap(a -= t * b, b), std::swap(u -= t * v, v);
96     }
97     if (u < 0) u += mod;
98     return u;
99 }
100 constexpr unsigned mod_pow_constexpr(ull a, ull n, unsigned mod) {
101     a %= mod;
102     ull res = 1;
103     for (int _ = 0; _ < 32; ++_) {
104         if (n & 1) res = res * a % mod;
105         a = a * a % mod, n /= 2;
106     }
107     return res;
108 }
109 unsigned mod_pow(ull a, ull n, unsigned mod) {
110     a %= mod;
111     ull res = 1;
112     for (int _ = 0; _ < 32; ++_) {
113         if (n & 1) res = res * a % mod;
114         a = a * a % mod, n /= 2;
115     }
116     return res;
117 }
118 ull mod_pow_64(ull a, ull n, ull mod) {
119     a %= mod;

```

```
120     ull res = 1;
121     while (n) {
122         if (n & 1) res = u128(res * a) % mod;
123         a = u128(a * a) % mod, n >>= 1;
124     }
125     return res;
126 }
127
128 template <typename T, unsigned p0, unsigned p1, unsigned p2>
129 T CRT3(ull a0, ull a1, ull a2) {
130     static_assert(p0 < p1 && p1 < p2);
131     static constexpr ull x0_1 = mod_pow_constexpr(p0, p1 - 2, p1);
132     static constexpr ull x01_2 = mod_pow_constexpr(ull(p0) * p1 % p2, p2 - 2, p2);
133     ull c = (a1 - a0 + p1) * x0_1 % p1;
134     ull a = a0 + c * p0;
135     c = (a2 - a % p2 + p2) * x01_2 % p2;
136     return T(a) + T(c) * T(p0) * T(p1);
137 }
138
139 template <typename mint>
140 mint inv(int n) {
141     static const int mod = mint::get_mod();
142     static vector<mint> dat = {0, 1};
143     assert(0 <= n);
144     if (n >= mod) n %= mod;
145     while (int(dat.size()) <= n) {
146         int k = dat.size();
147         auto q = (mod + k - 1) / k;
148         int r = k * q - mod;
149         dat.emplace_back(dat[r] * mint(q));
150     }
151     return dat[n];
152 }
153
154 template <typename mint>
155 mint fact(int n) {
156     static const int mod = mint::get_mod();
157     static vector<mint> dat = {1, 1};
158     assert(0 <= n);
159     if (n >= mod) return 0;
160     while (int(dat.size()) <= n) {
161         int k = dat.size();
162         dat.emplace_back(dat[k - 1] * mint(k));
163     }
164     return dat[n];
165 }
166
167 template <typename mint>
168 mint fact_inv(int n) {
169     static const int mod = mint::get_mod();
170     static vector<mint> dat = {1, 1};
171     assert(-1 <= n && n < mod);
172     if (n == -1) return mint(0);
173     while (int(dat.size()) <= n) {
174         int k = dat.size();
175         dat.emplace_back(dat[k - 1] * inv<mint>(k));
176     }
177     return dat[n];
178 }
179
180 template <typename mint>
181 mint C(ll n, ll m) {
182     if (m < 0 or m > n) return 0ll;
183     return fact<mint>(n) * fact_inv<mint>(m) * fact_inv<mint>(n - m);
184 }
```


others

快速取模

```
1 inline unsigned long long calc(const unsigned long long &x) {
2     return x - (__uint128_t(x) * 9920937979283557439ull >> 93) * 998244353;
3 }
```

date time

```
1 struct DateTime {
2     static constexpr int month_days[13]
3         = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
4     int year, month, day;
5     DateTime(int y, int m, int d) : year(y), month(m), day(d) {}
6
7     // 1年1月1日が 0 となるように変換
8     int to_int() {
9         int y = (month <= 2 ? year - 1 : year);
10        int m = (month <= 2 ? month + 12 : month);
11        int d = day;
12        return 365 * y + y / 4 - y / 100 + y / 400 + 306 * (m + 1) / 10 + d - 429;
13    }
14
15    // to_int() の逆関数
16    static DateTime from_int(int x) {
17        int y = x * 400 / 146097 + 1;
18        int d = x - DateTime(y, 1, 1).to_int();
19        int m = 1;
20        while (d >= 28) {
21            int k = month_days[m] + (m == 2 && is_leap_year(y) ? 1 : 0);
22            if (d < k) break;
23            ++m;
24            d -= k;
25        }
26        if (m == 13) {
27            ++y;
28            m = 1;
29        }
30        ++d;
31        return DateTime(y, m, d);
32    }
33
34    // 日曜日が 0 として、曜日を [0, 7) で返す
35    int weekday() { return (to_int() + 1) % 7; }
36
37    DateTime& operator++() {
38        ++day;
39        int lim = month_days[month];
40        if (is_leap_year(year) && month == 2) lim = 29;
41        if (day <= lim) return (*this);
42        day = 1;
43        ++month;
44        if (month == 13) {
45            ++year;
46            month = 1;
47        }
48        return (*this);
49    }
50    DateTime operator++(int) {
51        DateTime tmp = *this;
```



```
52     ++*this;
53     return tmp;
54 }
55
56 bool operator==(DateTime const& rhs) const {
57     return to_tuple() == rhs.to_tuple();
58 }
59 bool operator!=(DateTime const& rhs) const {
60     return to_tuple() != rhs.to_tuple();
61 }
62 bool operator<(DateTime const& rhs) const {
63     return to_tuple() < rhs.to_tuple();
64 }
65 bool operator<=(DateTime const& rhs) const {
66     return to_tuple() <= rhs.to_tuple();
67 }
68 bool operator>(DateTime const& rhs) const {
69     return to_tuple() > rhs.to_tuple();
70 }
71 bool operator>=(DateTime const& rhs) const {
72     return to_tuple() >= rhs.to_tuple();
73 }
74
75 // yyyy[sep]mm[sep]dd
76 string to_string(string sep = "-") {
77     string y = std::to_string(year);
78     string m = std::to_string(month);
79     string d = std::to_string(day);
80     while (len(y) < 4) y = "0" + y;
81     while (len(m) < 2) m = "0" + m;
82     while (len(d) < 2) d = "0" + d;
83     return y + sep + m + sep + d;
84 }
85
86 tuple<int, int, int> to_tuple() const { return {year, month, day}; }
87
88 static bool is_leap_year(int y) {
89     if (y % 400 == 0) return true;
90     return (y % 4 == 0 && y % 100 != 0);
91 }
92
93 static bool is_valid_date(int y, int m, int d) {
94     if (!(1 <= m && m <= 12)) return 0;
95     int mx = month_days[m];
96     if (m == 2 && is_leap_year(y)) ++mx;
97     return (1 <= d && d <= mx);
98 }
99 };
```