

Contents

1	Misc	3
1.1	Contest	3
1.1.1	Makefile	3
1.2	How Did We Get Here?	3
1.2.1	Macros	3
1.2.2	Fast I/O	3
1.2.3	constexpr	5
1.2.4	Bump Allocator	5
1.3	Tools	6
1.3.1	Floating Point Binary Search	6
1.3.2	SplitMix64	6
1.3.3	<random>	6
1.3.4	x86 Stack Hack	6
1.4	Algorithms	7
1.4.1	Bit Hacks	7
1.4.2	Infinite Grid Knight Distance	7
1.4.3	Longest Increasing Subsequence	7
1.4.4	Mo's Algorithm on Tree	7
2	Data Structures	9
2.1	GNU PBDS	9
2.2	2D Partial Sums	9

2.3	Heavy-Light Decomposition	10
3	Graph	12
3.1	Matching/Flows	12
3.1.1	Kuhn-Munkres algorithm	12
3.2	Shortest Path Faster Algorithm	13
3.3	Strongly Connected Components	14
3.4	Biconnected Components	14
3.4.1	Articulation Points	14
3.4.2	Bridges	15
3.5	Centroid Decomposition	15
4	Math	17
4.1	Number Theory	17
4.1.1	Mod Struct	17
4.1.2	Miller-Rabin	17
4.1.3	Linear Sieve	18
4.1.4	Get Factors	18
4.1.5	Binary GCD	19
4.1.6	Extended GCD	19
4.1.7	Chinese Remainder Theorem	19
4.1.8	Pollard's Rho	19
4.1.9	Rational Number Binary Search	20

4.1.10	De Brujin Sequence	20	6.5	Aho-Corasick Automaton	32
4.1.11	Long Long Multiplication	21			
5	Geometry	22	7	Debug List	34
5.1	Point	22			
5.1.1	Quarternion	22			
5.1.2	Spherical Coordinates	23			
5.2	Segments	24			
5.3	Convex Hull	24			
5.3.1	3D Hull	24			
5.4	Angular Sort	25			
5.5	Convex Polygon Minkowski Sum	26			
5.6	Point In Polygon	26			
5.6.1	Convex Version	26			
5.6.2	Offline Multiple Points Version	27			
5.7	Closest Pair	29			
5.8	Minimum Enclosing Circle	30			
6	Strings	31			
6.1	Knuth-Morris-Pratt Algorithm	31			
6.2	Z Value	31			
6.3	Manacher's Algorithm	31			
6.4	Minimum Rotation	32			

1. Misc

1.1. Contest

1.1.1. Makefile

```

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

```

1.2. How Did We Get Here?

1.2.1. Macros

Use vectorizations and math optimizations at your own peril.

For gcc \geq 9, there are `[[likely]]` and `[[unlikely]]` attributes.

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

```

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

```

1.2.2. Fast I/O

```

1 struct scanner {
   static constexpr size_t LEN = 32 << 20;
3 char *buf, *buf_ptr, *buf_end;
   scanner()
5     : buf(new char[LEN]), buf_ptr(buf + LEN),
       buf_end(buf + LEN) {}
7 ~scanner() { delete[] buf; }
   char getc() {
9     if (buf_ptr == buf_end) [[unlikely]]
       buf_end = buf + fread_unlocked(buf, 1, LEN, stdin),
11      buf_ptr = buf;
       return *(buf_ptr++);
13 }
   char seek(char del) {
15     char c;
       while ((c = getc()) < del) {}
17     return c;
   }
19 void read(int &t) {
       bool neg = false;
21     char c = seek('-');
       if (c == '-') neg = true, t = 0;
23     else t = c ^ '0';
       while ((c = getc()) >= '0') t = t * 10 + (c ^ '0');
25     if (neg) t = -t;
   }

```

```

27 };
28 struct printer {
29     static constexpr size_t CPI = 21, LEN = 32 << 20;
30     char *buf, *buf_ptr, *buf_end, *tbuf;
31     char *int_buf, *int_buf_end;
32     printer()
33         : buf(new char[LEN]), buf_ptr(buf),
34           buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
35           int_buf_end(int_buf + CPI - 1) {}
36     ~printer() {
37         flush();
38         delete[] buf, delete[] int_buf;
39     }
40     void flush() {
41         fwrite_unlocked(buf, 1, buf_ptr - buf, stdout);
42         buf_ptr = buf;
43     }
44     void write_(const char &c) {
45         *buf_ptr = c;
46         if (++buf_ptr == buf_end) [[unlikely]]
47             flush();
48     }
49     void write_(const char *s) {
50         for (; *s != '\0'; ++s) write_(*s);
51     }
52     void write(int x) {
53         if (x < 0) write_('-'), x = -x;

```

```

54         if (x == 0) [[unlikely]]
55             return write_('0');
56         for (tbuf = int_buf_end; x != 0; --tbuf, x /= 10)
57             *tbuf = '0' + char(x % 10);
58         write_(++tbuf);
59     }
60 };

```

Kotlin

```

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

```

```

    cout.println("omg hi")
19  cout.flush()
    }

```

1.2.3. constexpr

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

- constexpr recursion depth: 512
- constexpr loop iteration per function: 262 144
- constexpr operation count per function: 33 554 432
- template recursion depth: 900 (gcc *might* segfault first)

```

1  constexpr array<int, 10> fibonacci{[] {
    array<int, 10> a{};
3  a[0] = a[1] = 1;
    for (int i = 2; i < 10; i++) a[i] = a[i - 1] + a[i - 2];
5  return a;
    }()};
7  static_assert(fibonacci[9] == 55, "CE");

9  template <typename F, typename INT, INT... S>
constexpr void for_constexpr(integer_sequence<INT, S...>,
11                          F &&func) {
    int _[] = {(func(integral_constant<INT, S>{}), 0)...};
13 }
    // example
15 template <typename... T> void print_tuple(tuple<T...> t) {
    for_constexpr(make_index_sequence<sizeof...(T)>{}),
17     [&](auto i) { cout << get<i>(t) << '\n'; });

```

```

    }

```

1.2.4. Bump Allocator

```

1
3
    // global bump allocator
5  char mem[256 << 20]; // 256 MB
    size_t rsp = sizeof mem;
7  void *operator new(size_t s) {
    assert(s < rsp); // MLE
9  return (void *)&mem[rsp -= s];
    }
11 void operator delete(void *) {}

13 // bump allocator for STL / pbds containers
    char mem[256 << 20];
15 size_t rsp = sizeof mem;
    template <typename T> struct bump {
17     typedef T value_type;
        bump() {}
19     template <typename U> bump(U, ...) {}
        T *allocate(size_t n) {
21         rsp -= n * sizeof(T);
            rsp &= 0 - alignof(T);
23         return (T *) (mem + rsp);
        }
    }

```

```

25 void deallocate(T *, size_t n) {}
    };

```

1.3. Tools

1.3.1. Floating Point Binary Search

```

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

```

1.3.2. SplitMix64

```

1 using ull = unsigned long long;
    inline ull splitmix64(ull x) {
3 // change to `static ull x = SEED;` for DRBG
    ull z = (x += 0x9E3779B97F4A7C15);
5 z = (z ^ (z >> 30)) * 0xBF58476D1CE4E5B9;

```

```

    z = (z ^ (z >> 27)) * 0x94D049BB133111EB;
7 return z ^ (z >> 31);
    }

```

1.3.3. <random>

```

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

```

1.3.4. x86 Stack Hack

```

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

```

1.4. Algorithms

1.4.1. Bit Hacks

```

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

```

1.4.2. Infinite Grid Knight Distance

```

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

```

1.4.3. Longest Increasing Subsequence

```

1
3 template <class I> vi lis(const vector<I> &S) {
   if (S.empty()) return {};
5   vi prev(sz(S));

```

```

   typedef pair<I, int> p;
7   vector<p> res;
   rep(i, 0, sz(S)) {
9     // change 0 -> i for longest non-decreasing subsequence
     auto it = lower_bound(all(res), p{S[i], 0});
11    if (it == res.end())
       res.emplace_back(), it = res.end() - 1;
13    *it = {S[i], i};
     prev[i] = it == res.begin() ? 0 : (it - 1)->second;
15   }
   int L = sz(res), cur = res.back().second;
17   vi ans(L);
   while (L--) ans[L] = cur, cur = prev[cur];
19   return ans;
}

```

1.4.4. Mo's Algorithm on Tree

```

1 void MoAlgoOnTree() {
   Dfs(0, -1);
3   vector<int> euler(tk);
   for (int i = 0; i < n; ++i) {
5     euler[tin[i]] = i;
     euler[tout[i]] = i;
7   }
   vector<int> l(q), r(q), qr(q), sp(q, -1);
9   for (int i = 0; i < q; ++i) {
     if (tin[u[i]] > tin[v[i]]) swap(u[i], v[i]);

```

```
11  int z = GetLCA(u[i], v[i]);
    sp[i] = z[i];
13  if (z == u) l[i] = tin[u[i]], r[i] = tin[v[i]];
    else l[i] = tout[u[i]], r[i] = tin[v[i]];
15  qr[i] = i;
    }
17  sort(qr.begin(), qr.end(), [&](int i, int j) {
    if (l[i] / kB == l[j] / kB) return r[i] < r[j];
19  return l[i] / kB < l[j] / kB;
    });
21  vector<bool> used(n);
    // Add(v): add/remove v to/from the path based on used[v]
23  for (int i = 0, tl = 0, tr = -1; i < q; ++i) {
    while (tl < l[qr[i]]) Add(euler[tl++]);
25  while (tl > l[qr[i]]) Add(euler[--tl]);
    while (tr > r[qr[i]]) Add(euler[tr--]);
27  while (tr < r[qr[i]]) Add(euler[++tr]);
    // add/remove LCA(u, v) if necessary
29  }
    }
```

2. Data Structures

2.1. GNU PBDS

```

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

```

2.2. 2D Partial Sums

```

1 using vvi = vector<vector<int>>;

```

```

  using vvll = vector<vector<ll>>;
3 using vll = vector<ll>;
5 struct PrefixSum2D {
  vvll pref;           // 0-based 2-D prefix sum
7 void build(const vvll &v) { // creates a copy
  int n = v.size(), m = v[0].size();
9  pref.assign(n, vll(m, 0));
  for (int i = 0; i < n; i++) {
11     for (int j = 0; j < m; j++) {
        pref[i][j] = v[i][j] + (i ? pref[i - 1][j] : 0) +
13             (j ? pref[i][j - 1] : 0) -
                (i && j ? pref[i - 1][j - 1] : 0);
15     }
  }
17 }
  ll query(int ulx, int uly, int brx, int bry) const {
19     ll ans = pref[brx][bry];
    if (ulx) ans -= pref[ulx - 1][bry];
21     if (uly) ans -= pref[brx][uly - 1];
    if (ulx && uly) ans += pref[ulx - 1][uly - 1];
23     return ans;
  }
25 ll query(int ulx, int uly, int size) const {
    return query(ulx, uly, ulx + size - 1, uly + size - 1);
27 }
};

```

```

29 struct PartialSum2D : PrefixSum2D {
    vll diff; // 0 based
31 int n, m;
    PartialSum2D(int _n, int _m) : n(_n), m(_m) {
33     diff.assign(n + 1, vll(m + 1, 0));
    }
35 // add c from [ulx,uly] to [brx,bry]
    void update(int ulx, int uly, int brx, int bry, ll c) {
37     diff[ulx][uly] += c;
        diff[ulx][bry + 1] -= c;
39     diff[brx + 1][uly] -= c;
        diff[brx + 1][bry + 1] += c;
41     }
    void update(int ulx, int uly, int size, ll c) {
43     int brx = ulx + size - 1;
        int bry = uly + size - 1;
45     update(ulx, uly, brx, bry, c);
    }
47 // process the grid using prefix sum
    void process() { this->build(diff); }
49 };
// usage
51 PrefixSum2D pref;
    pref.build(v); // takes 2d 0-based vector as input
53 pref.query(x1, y1, x2, y2); // sum of region
55 PartialSum2D part(n, m); // dimension of grid 0 based

```

```

    part.update(x1, y1, x2, y2, 1); // add 1 in region
57 // must run after all updates
    part.process(); // prefix sum on diff array
59 // only exists after processing
    vll &grid = part.pref; // processed diff array
61 part.query(x1, y1, x2, y2); // gives sum of region

```

2.3. Heavy-Light Decomposition

```

1
3 template <bool VALS_EDGES> struct HLD {
    int N, tim = 0;
5     vector<vi> adj;
    vi par, siz, depth, rt, pos;
7     Node *tree;
    HLD(vector<vi> adj_)
9         : N(sz(adj_)), adj(adj_), par(N, -1), siz(N, 1),
            depth(N), rt(N), pos(N), tree(new Node(0, N)) {
11     dfsSz(0);
        dfsHld(0);
13     }
    void dfsSz(int v) {
15     if (par[v] != -1)
        adj[v].erase(find(all(adj[v]), par[v]));
17     for (int &u : adj[v]) {
        par[u] = v, depth[u] = depth[v] + 1;
19     dfsSz(u);
    }

```

```

    siz[v] += siz[u];
21   if (siz[u] > siz[adj[v][0]]) swap(u, adj[v][0]);
    }
23 }
void dfsHld(int v) {
25   pos[v] = tim++;
    for (int u : adj[v]) {
27       rt[u] = (u == adj[v][0] ? rt[v] : u);
        dfsHld(u);
29     }
    }
31 template <class B> void process(int u, int v, B op) {
    for (; rt[u] != rt[v]; v = par[rt[v]]) {
33         if (depth[rt[u]] > depth[rt[v]]) swap(u, v);
        op(pos[rt[v]], pos[v] + 1);
35     }
    if (depth[u] > depth[v]) swap(u, v);
37   op(pos[u] + VALS_EDGES, pos[v] + 1);
    }
39   void modifyPath(int u, int v, int val) {
        process(u, v,
41             [&](int l, int r) { tree->add(l, r, val); });
    }
43   int queryPath(int u,
        int v) { // Modify depending on problem
45   int res = -1e9;
        process(u, v, [&](int l, int r) {

```

```

47     res = max(res, tree->query(l, r));
        });
49     return res;
    }
51   int querySubtree(int v) { // modifySubtree is similar
        return tree->query(pos[v] + VALS_EDGES,
53             pos[v] + siz[v]);
    }
55 };

```

3.1.1. Kuhn-Munkres algorithm

25
27
29
31
33
35
37
39
41
43
45
47
49

25
27
29
31
33
35
37
39
41
43
45
47
49

```

51     for (int j = 0; j < n; j++) {
52         if (vx[j]) lx[j] -= d;
53         if (vy[j]) ly[j] += d;
54         else slack[j] -= d;
55     }
56 }
57 }
58 ll res = 0;
59 for (int i = 0; i < n; i++) {
60     res += edge[match[i]][i];
61 }
62 return res;
63 }
} graph;

```

3.2. Shortest Path Faster Algorithm

```

1 struct SPFA {
2     static const int maxn = 1010, INF = 1e9;
3     int dis[maxn];
4     bitset<maxn> inq, inneg;
5     queue<int> q, tq;
6     vector<pii> v[maxn];
7     void make_edge(int s, int t, int w) {
8         v[s].emplace_back(t, w);
9     }
10    void dfs(int a) {
11        inneg[a] = 1;

```

```

12    for (pii i : v[a])
13        if (!inneg[i.F]) dfs(i.F);
14    }
15    bool solve(int n, int s) { // true if have neg-cycle
16        for (int i = 0; i <= n; i++) dis[i] = INF;
17        dis[s] = 0, q.push(s);
18        for (int i = 0; i < n; i++) {
19            inq.reset();
20            int now;
21            while (!q.empty()) {
22                now = q.front(), q.pop();
23                for (pii &i : v[now]) {
24                    if (dis[i.F] > dis[now] + i.S) {
25                        dis[i.F] = dis[now] + i.S;
26                        if (!inq[i.F]) tq.push(i.F), inq[i.F] = 1;
27                    }
28                }
29            }
30            q.swap(tq);
31        }
32        bool re = !q.empty();
33        inneg.reset();
34        while (!q.empty()) {
35            if (!inneg[q.front()]) dfs(q.front());
36            q.pop();
37        }
38        return re;

```

```

39 }
    void reset(int n) {
41     for (int i = 0; i <= n; i++) v[i].clear();
    }
43 };

```

3.3. Strongly Connected Components

```

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

```

```

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

```

3.4. Biconnected Components

3.4.1. Articulation Points

```

1 void dfs(int x, int p) {
    tin[x] = low[x] = ++t;
3     int ch = 0;
    for (auto u : g[x])
5         if (u.first != p) {
            if (!ins[u.second])
7                 st.push(u.second), ins[u.second] = true;
            if (tin[u.first]) {
9                 low[x] = min(low[x], tin[u.first]);
            }
        }
}

```

```

11     continue;
12 }
13 ++ch;
14 dfs(u.first, x);
15 low[x] = min(low[x], low[u.first]);
16 if (low[u.first] >= tin[x]) {
17     cut[x] = true;
18     ++sz;
19     while (true) {
20         int e = st.top();
21         st.pop();
22         bcc[e] = sz;
23         if (e == u.second) break;
24     }
25 }
26 if (ch == 1 && p == -1) cut[x] = false;
27 }

```

3.4.2. Bridges

```

1 // if there are multi-edges, then they are not bridges
2 void dfs(int x, int p) {
3     tin[x] = low[x] = ++t;
4     st.push(x);
5     for (auto u : g[x])
6         if (u.first != p) {
7             if (tin[u.first]) {

```

```

9         low[x] = min(low[x], tin[u.first]);
10        continue;
11    }
12    dfs(u.first, x);
13    low[x] = min(low[x], low[u.first]);
14    if (low[u.first] == tin[u.first]) br[u.second] = true;
15    }
16    if (tin[x] == low[x]) {
17        ++sz;
18        while (st.size()) {
19            int u = st.top();
20            st.pop();
21            bcc[u] = sz;
22            if (u == x) break;
23        }
24    }
25 }

```

3.5. Centroid Decomposition

```

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

```

```
9    mx[now] = max(mx[now], sz[u]);
    sz[now] += sz[u];
11 }
}
13 void get_dis(int now, int d, int len) {
    dis[d][now] = cnt;
15 v[now] = true;
    for (auto u : G[now])
17     if (!v[u.first]) { get_dis(u, d, len + u.second); }
}
19 void dfs(int now, int fa, int d) {
    get_center(now);
21 int c = -1;
    for (int i : vtx) {
23         if (max(mx[i], (int)vtx.size() - sz[i]) <=
            (int)vtx.size() / 2)
25             c = i;
        v[i] = false;
27     }
    get_dis(c, d, 0);
29     for (int i : vtx) v[i] = false;
    v[c] = true;
31     vtx.clear();
    dep[c] = d;
33     p[c] = fa;
    for (auto u : G[c])
35         if (u.first != fa && !v[u.first]) {
```

```
37     dfs(u.first, c, d + 1);
    }
}
```

4. Math

4.1. Number Theory

4.1.1. Mod Struct

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

NTT prime p	$p - 1$	primitive root
65537	$1 \ll 16$	3
998244353	$119 \ll 23$	3
2748779069441	$5 \ll 39$	3
1945555039024054273	$27 \ll 56$	5

Requires: Extended GCD

```

1
3 template <typename T> struct M {
4     static T MOD; // change to constexpr if already known
5     T v;
6     M(T x = 0) {
7         v = (-MOD <= x && x < MOD) ? x : x % MOD;
8         if (v < 0) v += MOD;
9     }
10    explicit operator T() const { return v; }
11    bool operator==(const M &b) const { return v == b.v; }

```

```

12    bool operator!=(const M &b) const { return v != b.v; }
13    M operator-() { return M(-v); }
14    M operator+(M b) { return M(v + b.v); }
15    M operator-(M b) { return M(v - b.v); }
16    M operator*(M b) { return M((__int128)v * b.v % MOD); }
17    M operator/(M b) { return *this * (b ^ (MOD - 2)); }
18    // change above implementation to this if MOD is not prime
19    M inv() {
20        auto [p, _, g] = extgcd(v, MOD);
21        return assert(g == 1), p;
22    }
23    friend M operator^(M a, ll b) {
24        M ans(1);
25        for (; b >= 1, a *= a)
26            if (b & 1) ans *= a;
27        return ans;
28    }
29    friend M &operator+=(M &a, M b) { return a = a + b; }
30    friend M &operator-=(M &a, M b) { return a = a - b; }
31    friend M &operator*=(M &a, M b) { return a = a * b; }
32    friend M &operator/=(M &a, M b) { return a = a / b; }
33 };
34 using Mod = M<int>;
35 template <> int Mod::MOD = 1'000'000'007;
36 int &MOD = Mod::MOD;

```

4.1.2. Miller-Rabin

Requires: Mod Struct

```

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

```

4.1.3. Linear Sieve

```

1 constexpr ll MAXN = 1000000;
  bitset<MAXN> is_prime;
3  vector<ll> primes;
  ll mpf[MAXN], phi[MAXN], mu[MAXN];
5
void sieve() {
7   is_prime.set();
   is_prime[1] = 0;
9   mu[1] = phi[1] = 1;

```

```

11  for (ll i = 2; i < MAXN; i++) {
    if (is_prime[i]) {
13      mpf[i] = i;
      primes.push_back(i);
      phi[i] = i - 1;
15      mu[i] = -1;
    }
17    for (ll p : primes) {
      if (p > mpf[i] || i * p >= MAXN) break;
19      is_prime[i * p] = 0;
      mpf[i * p] = p;
21      mu[i * p] = -mu[i];
      if (i % p == 0)
23          phi[i * p] = phi[i] * p, mu[i * p] = 0;
      else phi[i * p] = phi[i] * (p - 1);
25    }
27 }

```

4.1.4. Get Factors

Requires: Linear Sieve

```

1
3 vector<ll> all_factors(ll n) {
  vector<ll> fac = {1};
5  while (n > 1) {
    const ll p = mpf[n];

```

```

7  vector<ll> cur = {1};
   while (n % p == 0) {
9      n /= p;
      cur.push_back(cur.back() * p);
11  }
   vector<ll> tmp;
13  for (auto x : fac)
      for (auto y : cur) tmp.push_back(x * y);
15  tmp.swap(fac);
   }
17  return fac;
   }

```

4.1.5. Binary GCD

```

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

```

4.1.6. Extended GCD

```

1  // returns (p, q, g): p * a + q * b == g == gcd(a, b)

```

```

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

```

4.1.7. Chinese Remainder Theorem

Requires: Extended GCD

```

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

```

4.1.8. Pollard's Rho

```

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

```

```

3 ll pollard_rho(ll n) {
  if (!(n & 1)) return 2;
5 while (1) {
  ll y = 2, x = RNG() % (n - 1) + 1, res = 1;
7   for (int sz = 2; res == 1; sz *= 2) {
     for (int i = 0; i < sz && res <= 1; i++) {
9       x = f(x, n);
       res = __gcd(abs(x - y), n);
11    }
    y = x;
13  }
  if (res != 0 && res != n) return res;
15 }
}

```

4.1.9. Rational Number Binary Search

```

1 struct QQ {
  ll p, q;
3  QQ go(QQ b, ll d) { return {p + b.p * d, q + b.q * d}; }
};
5 bool pred(QQ);
  // returns smallest p/q in [lo, hi] such that
7  // pred(p/q) is true, and 0 <= p,q <= N
  QQ frac_bs(ll N) {
9   QQ lo{0, 1}, hi{1, 0};
   if (pred(lo)) return lo;
11  assert(pred(hi));

```

```

  bool dir = 1, L = 1, H = 1;
13  for (; L || H; dir = !dir) {
    ll len = 0, step = 1;
15    for (int t = 0; t < 2 && (t ? step /= 2 : step *= 2);)
      if (QQ mid = hi.go(lo, len + step);
17         mid.p > N || mid.q > N || dir ^ pred(mid))
        t++;
19    else len += step;
    swap(lo, hi = hi.go(lo, len));
21    (dir ? L : H) = !!len;
  }
23  return dir ? hi : lo;
}

```

4.1.10. De Bruijn Sequence

```

1 int res[kN], aux[kN], a[kN], sz;
void Rec(int t, int p, int n, int k) {
3   if (t > n) {
     if (n % p == 0)
5     for (int i = 1; i <= p; ++i) res[sz++] = aux[i];
   } else {
7     aux[t] = aux[t - p];
     Rec(t + 1, p, n, k);
9     for (aux[t] = aux[t - p] + 1; aux[t] < k; ++aux[t])
       Rec(t + 1, t, n, k);
11  }
}

```

```
13 int DeBruijn(int k, int n) {  
    // return cyclic string of length k^n such that every  
15 // string of length n using k character appears as a  
    // substring.  
17 if (k == 1) return res[0] = 0, 1;  
    fill(aux, aux + k * n, 0);  
19 return sz = 0, Rec(1, 1, n, k), sz;  
}
```

4.1.11. Long Long Multiplication

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

5. Geometry

5.1. Point

```

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

```

5.1.1. Quaternion

```

1  constexpr double PI = 3.141592653589793;
    constexpr double EPS = 1e-7;
3  struct Q {
    using T = double;
5  T x, y, z, r;
    Q(T r = 0) : x(0), y(0), z(0), r(r) {}
7  Q(T x, T y, T z, T r = 0) : x(x), y(y), z(z), r(r) {}
    friend bool operator==(const Q &a, const Q &b) {
9      return (a - b).abs2() <= EPS;
    }
11 friend bool operator!=(const Q &a, const Q &b) {
    return !(a == b);
13 }
    Q operator-() { return Q(-x, -y, -z, -r); }
15 Q operator+(const Q &b) const {
    return Q(x + b.x, y + b.y, z + b.z, r + b.r);
17 }
    Q operator-(const Q &b) const {
19     return Q(x - b.x, y - b.y, z - b.z, r - b.r);
    }
21 Q operator*(const T &t) const {
    return Q(x * t, y * t, z * t, r * t);
23 }
    Q operator*(const Q &b) const {
25     return Q(r * b.x + x * b.r + y * b.z - z * b.y,
        r * b.y - x * b.z + y * b.r + z * b.x,

```

```

27     r * b.z + x * b.y - y * b.x + z * b.r,
        r * b.r - x * b.x - y * b.y - z * b.z);
29 }
Q operator/(const Q &b) const { return *this * b.inv(); }
31 T abs2() const { return r * r + x * x + y * y + z * z; }
T len() const { return sqrt(abs2()); }
33 Q conj() const { return Q(-x, -y, -z, r); }
Q unit() const { return *this * (1.0 / len()); }
35 Q inv() const { return conj() * (1.0 / abs2()); }
friend T dot(Q a, Q b) {
37     return a.x * b.x + a.y * b.y + a.z * b.z;
}
39 friend Q cross(Q a, Q b) {
    return Q(a.y * b.z - a.z * b.y, a.z * b.x - a.x * b.z,
41         a.x * b.y - a.y * b.x);
}
43 friend Q rotation_around(Q axis, T angle) {
    return axis.unit() * sin(angle / 2) + cos(angle / 2);
45 }
Q rotated_around(Q axis, T angle) {
47     Q u = rotation_around(axis, angle);
    return u * *this / u;
49 }
friend Q rotation_between(Q a, Q b) {
51     a = a.unit(), b = b.unit();
    if (a == -b) {
53         // degenerate case

```

```

    Q ortho = abs(a.y) > EPS ? cross(a, Q(1, 0, 0))
        : cross(a, Q(0, 1, 0));
55     return rotation_around(ortho, PI);
57 }
    return (a * (a + b)).conj();
59 }
};

```

5.1.2. Spherical Coordinates

```

1 struct car_p {
    double x, y, z;
3 };
struct sph_p {
5     double r, theta, phi;
};
7
sph_p conv(car_p p) {
9     double r = sqrt(p.x * p.x + p.y * p.y + p.z * p.z);
    double theta = asin(p.y / r);
11    double phi = atan2(p.y, p.x);
    return {r, theta, phi};
13 }
car_p conv(sph_p p) {
15    double x = p.r * cos(p.theta) * sin(p.phi);
    double y = p.r * cos(p.theta) * cos(p.phi);
17    double z = p.r * sin(p.theta);
    return {x, y, z};

```

19 }

5.2. Segments

```

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

```

5.3. Convex Hull

```

1 // returns a convex hull in counterclockwise order
  // for a non-strict one, change cross >= to >
3 vector<pt> convex_hull(vector<pt> p) {
   sort(ALL(p));
5   if (p[0] == p.back()) return {p[0]};
   int n = p.size(), t = 0;

```

```

7   vector<pt> h(n + 1);
   for (int _ = 2, s = 0; _--; s = --t, reverse(ALL(p)))
9     for (pt i : p) {
        while (t > s + 1 && cross(i, h[t - 1], h[t - 2]) >= 0)
11        t--;
        h[t++] = i;
13    }
   return h.resize(t), h;
15 }

```

5.3.1. 3D Hull

```

1
3 typedef Point3D<double> P3;
5 struct PR {
   void ins(int x) { (a == -1 ? a : b) = x; }
7   void rem(int x) { (a == x ? a : b) = -1; }
   int cnt() { return (a != -1) + (b != -1); }
9   int a, b;
};
11
13 struct F {
   P3 q;
   int a, b, c;
15 };

```

```

17 vector<F> hull3d(const vector<P3> &A) {
    assert(sz(A) >= 4);
19   vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1, -1}));
    #define E(x, y) E[f.x][f.y]
21   vector<F> FS;
    auto mf = [&](int i, int j, int k, int l) {
23       P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
        if (q.dot(A[l]) > q.dot(A[i])) q = q * -1;
25       F f{q, i, j, k};
        E(a, b).ins(k);
27       E(a, c).ins(j);
        E(b, c).ins(i);
29       FS.push_back(f);
    };
31   rep(i, 0, 4) rep(j, i + 1, 4) rep(k, j + 1, 4)
    mf(i, j, k, 6 - i - j - k);
33
    rep(i, 4, sz(A)) {
35       rep(j, 0, sz(FS)) {
        F f = FS[j];
37         if (f.q.dot(A[i]) > f.q.dot(A[f.a])) {
            E(a, b).rem(f.c);
39             E(a, c).rem(f.b);
            E(b, c).rem(f.a);
41             swap(FS[j--], FS.back());
            FS.pop_back();
43         }
    }

```

```

    }
45   int nw = sz(FS);
    rep(j, 0, nw) {
47       F f = FS[j];
        #define C(a, b, c)
49       if (E(a, b).cnt() != 2) mf(f.a, f.b, i, f.c);
        C(a, b, c);
51       C(a, c, b);
        C(b, c, a);
53     }
    }
55   for (F &it : FS)
        if ((A[it.b] - A[it.a])
57             .cross(A[it.c] - A[it.a])
             .dot(it.q) <= 0)
59       swap(it.c, it.b);
    return FS;
61 };

```

5.4. Angular Sort

```

1 auto angle_cmp = [](const pt &a, const pt &b) {
    auto btm = [](const pt &a) {
3       return a.y < 0 || (a.y == 0 && a.x < 0);
    };
5   return make_tuple(btm(a), a.y * b.x, abs2(a)) <
        make_tuple(btm(b), a.x * b.y, abs2(b));
7 };

```

```

void angular_sort(vector<pt> &p) {
9   sort(p.begin(), p.end(), angle_cmp);
}

```

5.5. Convex Polygon Minkowski Sum

```

1 // O(n) convex polygon minkowski sum
  // must be sorted and counterclockwise
3 vector<pt> minkowski_sum(vector<pt> p, vector<pt> q) {
  auto diff = [](vector<pt> &c) {
5     auto rcmp = [](pt a, pt b) {
        return pt{a.y, a.x} < pt{b.y, b.x};
7     };
    rotate(c.begin(), min_element(ALL(c), rcmp), c.end());
9     c.push_back(c[0]);
    vector<pt> ret;
11    for (int i = 1; i < c.size(); i++)
        ret.push_back(c[i] - c[i - 1]);
13    return ret;
  };
15  auto dp = diff(p), dq = diff(q);
  pt cur = p[0] + q[0];
17  vector<pt> d(dp.size() + dq.size()), ret = {cur};
  // include angle_cmp from angular-sort.cpp
19  merge(ALL(dp), ALL(dq), d.begin(), angle_cmp);
  // optional: make ret strictly convex (UB if degenerate)
21  int now = 0;
  for (int i = 1; i < d.size(); i++) {

```

```

23     if (cross(d[i], d[now]) == 0) d[now] = d[now] + d[i];
        else d[++now] = d[i];
25     }
    d.resize(now + 1);
27    // end optional part
    for (pt v : d) ret.push_back(cur = cur + v);
29    return ret.pop_back(), ret;
  }

```

5.6. Point In Polygon

```

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

```

5.6.1. Convex Version

```

1 // no preprocessing version

```

```

// p must be a strict convex hull, counterclockwise
3 // if point is inside or on border
bool is_inside(const vector<pt> &c, pt p) {
5   int n = c.size(), l = 1, r = n - 1;
   if (cross(c[0], c[1], p) < 0) return false;
7   if (cross(c[n - 1], c[0], p) < 0) return false;
   while (l < r - 1) {
9     int m = (l + r) / 2;
     T a = cross(c[0], c[m], p);
11    if (a > 0) l = m;
     else if (a < 0) r = m;
13    else return dot(c[0] - p, c[m] - p) <= 0;
   }
15   if (l == r) return dot(c[0] - p, c[l] - p) <= 0;
   else return cross(c[l], c[r], p) >= 0;
17 }

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

```

```

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

```

5.6.2. Offline Multiple Points Version

Requires: GNU PBDS, Point

```

1
3
5
using Double = __float128;
7 using Point = pt<Double, Double>;
9 int n, m;

```

```

vector<Point> poly;
11 vector<Point> query;
vector<int> ans;
13
struct Segment {
15     Point a, b;
    int id;
17 };
vector<Segment> segs;
19
Double Xnow;
21 inline Double get_y(const Segment &u, Double xnow = Xnow) {
    const Point &a = u.a;
23     const Point &b = u.b;
    return (a.y * (b.x - xnow) + b.y * (xnow - a.x)) /
25     (b.x - a.x);
}
27 bool operator<(Segment u, Segment v) {
    Double yu = get_y(u);
29     Double yv = get_y(v);
    if (yu != yv) return yu < yv;
31     return u.id < v.id;
}
33 ordered_map<Segment> st;

35 struct Event {
    int type; // +1 insert seg, -1 remove seg, 0 query

```

```

37     Double x, y;
    int id;
39 };
bool operator<(Event a, Event b) {
41     if (a.x != b.x) return a.x < b.x;
    if (a.type != b.type) return a.type < b.type;
43     return a.y < b.y;
}
45 vector<Event> events;

47 void solve() {
    set<Double> xs;
49     set<Point> ps;
    for (int i = 0; i < n; i++) {
51         xs.insert(poly[i].x);
        ps.insert(poly[i]);
53     }
    for (int i = 0; i < n; i++) {
55         Segment s{poly[i], poly[(i + 1) % n], i};
        if (s.a.x > s.b.x ||
57         (s.a.x == s.b.x && s.a.y > s.b.y)) {
            swap(s.a, s.b);
59         }
        segs.push_back(s);
61
        if (s.a.x != s.b.x) {
63             events.push_back({+1, s.a.x + 0.2, s.a.y, i});

```

```

    events.push_back({-1, s.b.x - 0.2, s.b.y, i});
65 }
    }
67 for (int i = 0; i < m; i++) {
    events.push_back({0, query[i].x, query[i].y, i});
69 }
    sort(events.begin(), events.end());
71 int cnt = 0;
    for (Event e : events) {
73     int i = e.id;
        Xnow = e.x;
75     if (e.type == 0) {
        Double x = e.x;
77         Double y = e.y;
        Segment tmp = {{x - 1, y}, {x + 1, y}, -1};
79         auto it = st.lower_bound(tmp);

81         if (ps.count(query[i]) > 0) {
            ans[i] = 0;
83         } else if (xs.count(x) > 0) {
            ans[i] = -2;
85         } else if (it != st.end() &&
                    get_y(*it) == get_y(tmp)) {
87             ans[i] = 0;
        } else if (it != st.begin() &&
89             get_y(*prev(it)) == get_y(tmp)) {
            ans[i] = 0;

```

```

91     } else {
        int rk = st.order_of_key(tmp);
93         if (rk % 2 == 1) {
            ans[i] = 1;
95         } else {
            ans[i] = -1;
97         }
        }
99     } else if (e.type == 1) {
        st.insert(segs[i]);
101         assert((int)st.size() == ++cnt);
        } else if (e.type == -1) {
103             st.erase(segs[i]);
            assert((int)st.size() == --cnt);
105         }
        }
107     }

```

5.7. Closest Pair

```

1 vector<pll> p; // sort by x first!
    bool cmpy(const pll &a, const pll &b) const {
3         return a.y < b.y;
    }
5 ll sq(ll x) { return x * x; }
    // returns (minimum dist)^2 in [l, r)
7 ll solve(int l, int r) {
    if (r - l <= 1) return 1e18;

```

```

9  int m = (l + r) / 2;
    ll mid = p[m].x, d = min(solve(l, m), solve(m, r));
11  auto pb = p.begin();
    inplace_merge(pb + l, pb + m, pb + r, cmpy);
13  vector<pll> s;
    for (int i = l; i < r; i++)
15      if (sq(p[i].x - mid) < d) s.push_back(p[i]);
    for (int i = 0; i < s.size(); i++)
17      for (int j = i + 1;
           j < s.size() && sq(s[j].y - s[i].y) < d; j++)
19        d = min(d, dis(s[i], s[j]));
    return d;
21 }

```

5.8. Minimum Enclosing Circle

```

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

```

```

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

```

6. Strings

6.1. Knuth-Morris-Pratt Algorithm

```

1 vector<int> pi(const string &s) {
  vector<int> p(s.size());
3  for (int i = 1; i < s.size(); i++) {
    int g = p[i - 1];
5    while (g && s[i] != s[g]) g = p[g - 1];
    p[i] = g + (s[i] == s[g]);
7  }
  return p;
9 }

vector<int> match(const string &s, const string &pat) {
11 vector<int> p = pi(pat + '\0' + s), res;
  for (int i = p.size() - s.size(); i < p.size(); i++)
13   if (p[i] == pat.size())
    res.push_back(i - 2 * pat.size());
15 return res;
}

```

6.2. Z Value

```

1 int z[n];
void zval(string s) {
3  // z[i] => longest common prefix of s and s[i:], i > 0
  int n = s.size();
5  z[0] = 0;
  for (int b = 0, i = 1; i < n; i++) {

```

```

7   if (z[b] + b <= i) z[i] = 0;
   else z[i] = min(z[i - b], z[b] + b - i);
9   while (s[i + z[i]] == s[z[i]]) z[i]++;
   if (i + z[i] > b + z[b]) b = i;
11  }
}

```

6.3. Manacher's Algorithm

```

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

```

6.4. Minimum Rotation

```
1 int min_rotation(string s) {
2     int a = 0, n = s.size();
3     s += s;
4     for (int b = 0; b < n; b++) {
5         for (int k = 0; k < n; k++) {
6             if (a + k == b || s[a + k] < s[b + k]) {
7                 b += max(0, k - 1);
8                 break;
9             }
10            if (s[a + k] > s[b + k]) {
11                a = b;
12                break;
13            }
14        }
15    }
16    return a;
17 }
```

6.5. Aho-Corasick Automaton

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

```

27     tmp = T[ptr].fail;
28     while (tmp != 1 && !T[tmp].Next[i])
29         tmp = T[tmp].fail;
30     if (T[tmp].Next[i] != T[ptr].Next[i])
31         if (T[tmp].Next[i]) tmp = T[tmp].Next[i];
32     T[T[ptr].Next[i]].fail = tmp;
33     q[qtop++] = T[ptr].Next[i];
34 }
35 }
36 void AC_auto(const string &s) {
37     int ptr = 1;
38     for (char c : s) {
39         while (ptr != 1 && !T[ptr].Next[c]) ptr = T[ptr].fail;
40         if (T[ptr].Next[c]) {
41             ptr = T[ptr].Next[c];
42             T[ptr].ans++;
43         }
44     }
45 }
46 void Solve(string &s) {
47     for (char &c : s) // change char id
48         c -= 'a';
49     for (int i = 0; i < qtop; i++) build_fail(q[i]);
50     AC_auto(s);
51     for (int i = qtop - 1; i > -1; i--)
52         T[T[q[i]].fail].ans += T[q[i]].ans;
53 }

```

```

54 void reset() {
55     qtop = top = q[0] = 1;
56     get_node(1);
57 }
58 } AC;
59 // usage example
60 string s, S;
61 int n, t, ans_place[50000];
62 int main() {
63     Tie cin >> t;
64     while (t--) {
65         AC.reset();
66         cin >> S >> n;
67         for (int i = 0; i < n; i++) {
68             cin >> s;
69             ans_place[i] = AC.insert(s);
70         }
71         AC.Solve(S);
72         for (int i = 0; i < n; i++)
73             cout << AC.T[ans_place[i]].ans << '\n';
74     }
75 }

```

7. Debug List

- | | |
|--|--|
| <p>1 - Pre-submit:</p> <p>3 - Did you make a typo when copying a template?</p> <p>3 - Test more cases if unsure.</p> <p>5 - Write a naive solution and check small cases.</p> <p>5 - Submit the correct file.</p> <p>7 - General Debugging:</p> <p>9 - Read the whole problem again.</p> <p>9 - Have a teammate read the problem.</p> <p>11 - Have a teammate read your code.</p> <p>11 - Explain your solution to them (or a rubber duck).</p> <p>13 - Print the code and its output / debug output.</p> <p>13 - Go to the toilet.</p> <p>15 - Wrong Answer:</p> <p>17 - Any possible overflows?</p> <p>17 - > <code>__int128</code> ?</p> <p>19 - Try <code>-ftapv</code> or <code>#pragma GCC optimize("trapv")</code></p> <p>19 - Floating point errors?</p> <p>21 - > <code>long double</code> ?</p> <p>21 - turn off math optimizations</p> <p>23 - check for <code>'=='</code>, <code>'>='</code>, <code>'acos(1.000000001)'</code>, etc.</p> <p>23 - Did you forget to sort or unique?</p> <p>25 - Generate large and worst "corner" cases.</p> <p>25 - Check your <code>'m' / 'n', 'i' / 'j'</code> and <code>'x' / 'y'</code>.</p> <p>- Are everything initialized or reset properly?</p> | <p>27 - Are you sure about the STL thing you are using?</p> <p>- Read <code>cppreference</code> (should be available).</p> <p>29 - Print everything and run it on pen and paper.</p> <p>31 - Time Limit Exceeded:</p> <p>- Calculate your time complexity again.</p> <p>33 - Does the program actually end?</p> <p>- Check for <code>'while(q.size())'</code> etc.</p> <p>35 - Test the largest cases locally.</p> <p>- Did you do unnecessary stuff?</p> <p>37 - e.g. pass vectors by value</p> <p>- e.g. <code>'memset'</code> for every test case</p> <p>39 - Is your constant factor reasonable?</p> <p>41 - Runtime Error:</p> <p>- Check memory usage.</p> <p>43 - Forget to clear or destroy stuff?</p> <p>- > <code>'vector::shrink_to_fit()'</code></p> <p>45 - Stack overflow?</p> <p>- Bad pointer / array access?</p> <p>47 - Try <code>'-fsanitize=address'</code></p> <p>- Division by zero? NaN's?</p> |
|--|--|