UNIVERSITEIT VAN AMSTERDAM

Monad Maniacs

```cpp
/* Union Find data structure. */
struct UF {
    vector<int> par, rank;
    int setc;
    UF(int n) : par(n), rank(n, 0), setc(n) {
        for (int i = 0; i < n; ++i) par[i] = i;
    }

    int root(int u) {
        int r = par[u];
        if (u == r) return r;
        return par[u] = root(r);
    }

    void unite(int u, int v) {
        int ru = root(u);
        int rv = root(v);
        if (ru == rv) return;
        --setc;
        if (rank[ru] < rank[rv]) {
            par[ru] = rv;
        } else if (rank[ru] > rank[rv]) {
            par[rv] = ru;
        } else {
            par[rv] = ru;
            ++rank[ru];
        }
    }

    bool same_set(int u, int v) {
        return root(u) == root(v);
    }
};
```

```cpp
/* Segment Tree data structure. */
template <typename T>
struct SegTree {
    int elementc;
    vector<T> tree;  // Root has index 1; the 0th element is unused.

    // CHANGE THIS SEGMENT
    const T id = 0;
    T operation(T a, T b) {
        return a + b;
    }
    // UNTIL HERE

    /* Build a segment tree with n elements and initializes them to id. */
    SegTree(int n) : elementc(n), tree(2 * n, id) {}

    /* Build a segment tree from arr. */
    SegTree(vector<T>& arr) : elementc(arr.size()), tree(2 * arr.size(), id) {
        copy(arr.begin(), arr.end(), tree.begin() + elementc);  // Copy the array values to the bottom of the tree
        for (int i = elementc - 1; i > 0; --i) tree[i] = operation(tree[2 * i], tree[2 * i + 1]);
    }

    /* Returns cumulative result of applying segtree.operation on [left, right). Returns segtree.identity if the
     * interval is empty. */
    T query(unsigned left, unsigned right) {
        T result = id;
        left += elementc;
        right += elementc;
        while (left < right) {
            if (left & 1) result = operation(result, tree[left++]);
            if (right & 1) result = operation(result, tree[--right]);
            left >>= 1;  // Dividing by 2
            right >>= 1;
        }
        return result;
    }

    /* Assigns value to element at index. */
    void assign(unsigned index, T value) {
        index += elementc;
        tree[index] = value;
        while (index > 1) {
            tree[index >> 1] = operation(tree[index], tree[index ^ 1]);
            index >>= 1;
        }
    }
```

```
};
```

```
/* Segment Tree (Roberts version). */
class SegTreeR {
    SegTreeR *left, *right;
    int from, to, value;

    SegTreeR(const vector<int>& arr, int l, int r) : left(NULL), right(NULL), from(l), to(r), value(0) {
        if (l == r)
            value = arr[l];
        else {
            int m = (l + r) / 2;
            left  = new SegTreeR(arr, l, m);
            right = new SegTreeR(arr, m + 1, r);
            value += left->value;
            value += right->value;
        }
    }

   public:
    SegTreeR(vector<int>& arr) {
        *this = SegTreeR(arr, 0, arr.size() - 1);
    }

    int sum(int l, int r) {
        if (l <= from && r >= to) return value;
        if (l > to || r < from) return 0;
        return left->sum(l, r) + right->sum(l, r);
    }

    int update(int i, int val) {
        if (i > to || i < from) return value;
        if (from == to)
            value = val;
        else
            value = left->update(i, val) + right->update(i, val);
        return value;
    }

    ~SegTreeR() {
        delete left;
        delete right;
    }
};
```

```
/* Performs binary search on a and returns the index of t, or -1 if not found. */
unsigned long long binsrch(vector<unsigned long long>& a, unsigned long long t) {
    unsigned long long l = 0, r = a.size() - 1, m;
    while (l <= r) {
        m = l + (r - l) / 2;
        if (a[m] == t)
            return m;
        else if (a[m] < t)
            l = m + 1;
        else
            r = m - 1;
    }
    return -1;
}
```

```
void dfs(vector<vector<int>>& adj, int src) {
    stack<int> st;
    st.push(src);
    vector<bool> seen(adj.size(), false);
    seen[src] = true;
    int u;
    while (!st.empty()) {
        u = st.top();
        st.pop();
        for (auto v : adj[u]) {
            if (!seen[v]) {
                seen[v] = true;
                st.push(v);
            }
        }
    }
}
```

```
void bfs(vector<vector<int>>& adj, int src) {
```

```
    queue<int> q;
    q.push(src);
    vector<int> dist(adj.size(), -1);
    dist[src] = 0;
    int u;
    while (!q.empty()) {
        u = q.front();
        q.pop();
        for (auto v : adj[u]) {
            if (dist[v] == -1) {
                dist[v] = dist[u] + 1;
                q.push(v);
            }
        }
    }
}
```

```
bool bipartite(vector<vector<int>>& adj) {
    auto N = adj.size();
    vector<int> color(N, 0);
    stack<int> st;
    for (auto src = 0; src < N; ++src) {
        if (color[src]) continue;
        st.push(src);
        color[src] = 1;
        while (st.size()) {
            int u = st.top();
            st.pop();
            for (auto v : adj[u]) {
                if (!color[v]) {
                    color[v] = 3 - color[u];
                    st.push(v);
                } else if (color[v] == color[u]) {
                    return false;
                }
            }
        }
    }
    return true;
}
```

```
/* Computes the weight of the minimal spanning tree. Format of edges[i] is { weight, from, to }. */
int kruskal(vector<vector<int>>& edges, int n) {
    sort(edges.begin(), edges.end());
    int weight = 0;
    UF uf      = UF(n);
    for (auto& edge : edges) {
        if (!uf.same_set(edge[1], edge[2])) {
            uf.unite(edge[1], edge[2]);
            weight += edge[0];
        }
    }
    return weight;
}
```

```
/* Toposort khan. */
vector<int> toposort(int n, vector<vector<int>>& adj) {
    vector<int> res, stack, p(n);
    for (int u = 0; u < n; ++u)
        for (int child : adj[u]) ++p[child];
    for (int u = 0; u < n; ++u)
        if (p[u] == 0) stack.push_back(u);
    while (!stack.empty()) {
        int u = stack.back();
        stack.pop_back();
        res.push_back(u);
        for (int v : adj[u]) {
            --p[v];
            if (p[v] == 0) stack.push_back(v);
        }
    }
    if (res.size() < n) return {-1};
    return res;
}
```

```
/* Returns the length of the shortest path from start to every node, or -1 if no path to that node exists.
    adj[a] contains a list of pairs { b, w }, meaning a is adjacent to b with weight w */
vector<int> dijkstra(vector<vector<pair<int, int>>>& adj, int src) {
    vector<int> dist(adj.size(), INT_MAX);                              // distance
```

```
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<>> q;  // { distance, node }
    dist[src] = 0;
    q.emplace(0, src);
    while (!q.empty()) {
        auto [du, u] = q.top();
        q.pop();
        if (du != dist[u]) continue;
        for (auto [v, w] : adj[u]) {
            if (dist[v] > dist[u] + w) {
                dist[v] = dist[u] + w;
                q.emplace(dist[v], v);
            }
        }
    }
    return dist;
}
```

```
/* Returns shortest distance from node s to node i. Format of node is adj[from][i] = { to, weight }. */
vector<int> bellman_ford(vector<vector<pair<int, int>>>& adj, int s) {
    int n = adj.size();
    vector<int> dist(n, INT_MAX / 2);
    dist[s] = 0;
    for (int i = 0; i < n - 1; ++i)
        for (int u = 0; u < n; ++u)
            for (auto [v, w] : adj[u]) dist[v] = min(dist[v], dist[u] + w);
    return dist;
}
```

```
/* Returns shortest distance from node i to node j. Takes a matix of weights as input. s*/
vector<vector<int>> floyd_warshall(vector<vector<int>>& weight) {
    int n                    = weight.size();
    vector<vector<int>> dist = weight;
    for (int k = 0; k < n; ++k)
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j) dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
    return dist;
}
```

```
/* Computes maximum flow using an implementation of the Edmonds-Karp algorithm. */
template <typename T>
T max_flow(vector<vector<int>>& adj, vector<vector<T>> capacity, int source, int sink) {
    const T INFTY = numeric_limits<T>::max();
    T flow        = 0;
    vector<int> parent(adj.size());
    while (true) {
        fill(parent.begin(), parent.end(), -1);
        parent[source] = -2;
        queue<pair<int, T>> q;
        q.emplace(source, INFTY);
        bool found_path = false;
        T current_flow  = INFTY;
        while (!q.empty() && !found_path) {
            int current_node = q.front().first;
            current_flow     = q.front().second;
            q.pop();
            for (auto next_node : adj[current_node]) {
                if (parent[next_node] == -1 && capacity[current_node][next_node]) {
                    parent[next_node] = current_node;
                    current_flow      = min(current_flow, capacity[current_node][next_node]);
                    if (next_node == sink) {
                        found_path = true;
                        break;
                    }
                    q.emplace(next_node, current_flow);
                }
            }
        }
        if (!found_path) break;
        flow += current_flow;
        int current_node = sink, previous_node;
        while (current_node != source) {
            previous_node = parent[current_node];
            capacity[previous_node][current_node] -= current_flow;
            capacity[current_node][previous_node] += current_flow;
            current_node = previous_node;
        }
    }
    return flow;
```

```
}
```

```
/* Tarjan SCC. Initialization:
u = curnum = compc = 0; num = lowln = comp {-1}; stack = {};
*/
void SCC(int u, vector<vector<int>>& adj, int& curnum, int& compc, vector<int>& num, vector<int>& lowln,
         vector<int>& comp, vector<int>& stack) {
    stack.push_back(u);
    lowln[u] = num[u] = curnum++;
    for (int v : adj[u])
        if (num[v] == -1) {
            SCC(v, adj, curnum, compc, num, lowln, comp, stack);
            lowln[u] = min(lowln[u], lowln[v]);
        } else if (comp[v] == -1)
            lowln[u] = min(lowln[u], lowln[v]);
    if (num[u] == lowln[u]) {
        for (int v = -1; v != u;) {
            v = stack.back();
            stack.pop_back();
            comp[v] = compc;
        }
        compc++;
    }
}
```

```
int KMP(const string& s, const string& t) {
    int m = t.size();
    vector<int> pi(m + 1);
    pi[0] = 0;
    if (m) pi[1] = 0;
    for (int i = 2; i <= m; ++i) {
        for (int j = pi[i - 1];; j = pi[j]) {
            if (t[j] == t[i - 1]) {
                pi[i] = j + 1;
                break;
            }
            if (j == 0) {
                pi[i] = 0;
                break;
            }
        }
    }
    int count = 0;
    int n     = s.size();
    for (int i = 0, j = 0; i < n;) {
        if (s[i] == t[j]) {
            ++i;
            ++j;
            if (j == m) {
                ++count;
                j = pi[j];
            }
        } else if (j > 0)
            j = pi[j];
        else
            ++i;
    }
    return count;
}
```

```
/* Tries. */
typedef struct Trie {
    Trie* child[26];
    bool end;
    bool has_child;
    Trie() : child{}, end(false), has_child(false) {}
} Trie;

void trie_insert(Trie* root, string& s) {
    for (auto c : s) {
        root->has_child = true;
        c -= 'a';
        if (!root->child[c]) root->child[c] = new Trie();
        root = root->child[c];
    }
    root->end = true;
}

void trie_delete(Trie* root) {
    for (Trie* child : root->child)
```

```
        if (child) trie_delete(child);
    delete root;
}

bool trie_contains(Trie* root, string& s) {
    for (auto c : s) {
        c -= 'a';
        if (!root->child[c]) return false;
        root = root->child[c];
    }
    return root->end;
}
```

```
/* Suffix array. */
struct suffix_array {
    struct entry {
        pair<int, int> nr;
        int p;
        bool operator<(const entry& other) const {
            return nr < other.nr;
        }
    };
    string s;
    int n;
    vector<vector<int>> P;
    vector<entry> L;
    vector<int> idx;

    suffix_array(string _s) : s(_s), n(s.size()) {
        L = vector<entry>(n);
        P.push_back(vector<int>(n));
        idx = vector<int>(n);
        for (int i = 0; i < n; i++) P[0][i] = s[i];
        for (int stp = 1, cnt = 1; (cnt >> 1) < n; stp++, cnt <<= 1) {
            P.push_back(vector<int>(n));
            for (int i = 0; i < n; i++) {
                L[i].p  = i;
                L[i].nr = make_pair(P[stp - 1][i], i + cnt < n ? P[stp - 1][i + cnt] : -1);
            }
            sort(L.begin(), L.end());
            for (int i = 0; i < n; i++)
                if (i > 0 && L[i].nr == L[i - 1].nr)
                    P[stp][L[i].p] = P[stp][L[i - 1].p];
                else
                    P[stp][L[i].p] = i;
        }
        for (int i = 0; i < n; i++) idx[P[P.size() - 1][i]] = i;
    }

    /* Longest common prefix. */
    int lcp(int x, int y) {
        int res = 0;
        if (x == y) return n - x;
        for (int k = P.size() - 1; k >= 0 && x < n && y < n; k--) {
            if (P[k][x] == P[k][y]) {
                x += 1 << k;
                y += 1 << k;
                res += 1 << k;
            }
        }
        return res;
    }
};
```

```
/* Computes gcd(a, b) and finds x, y such that ax + by = gcd(a, b). */
long long extgcd(long long a, long long b, long long& x, long long& y) {
    x = 1, y = 0;
    long long x1 = 0, y1 = 1, q, temp;
    while (b) {
        q    = a / b;
        temp = x;
        x    = x1;
        x1   = temp - q * x1;
        temp = y;
        y    = y1;
        y1   = temp - q * y1;
        temp = a;
        a    = b;
        b    = temp - q * b;
    }
    return a;
```

```
}
```

```
/* Returns the smallest non-negative x such that x ≡ r[i] mod m[i] for all i; assumes m[i] are pairwise coprime. */
long long crt(vector<long long>& r, vector<long long>& m) {
    long long crt = 0, N = 1, x, y;
    for (auto n : m) N *= n;
    for (int i = 0; i < m.size(); ++i) {
        extgcd(N / m[i], m[i], x, y);
        crt = (crt + N / m[i] * r[i] * x) % N;  // % N ensures crt is minimal.
    }
    return crt < 0 ? crt + N : crt;  // Ensure crt is positive.
}
```

```
/* Returns the smallest non-negative x such that x ≡ r[i] mod m[i] for all i; works with non-coprime m[i], returns -1 if
 * no solution exists. */
long long crt_npc(vector<long long>& r, vector<long long>& m) {
    long long crt = 0, N = 1, g, x, y;
    for (int i = 0; i < m.size(); ++i) {
        g = extgcd(N, m[i], x, y);
        if ((crt - r[i]) % g) return -1;  // No solution exists.
        N   = N / g * m[i];
        crt = (r[i] + m[i] / g * y * (crt - r[i])) % N;  // % N ensures crt is minimal.
    }
    return crt < 0 ? crt + N : crt;  // Ensure crt is positive.
}
```

```
/* Returns (a^b) mod m. */
unsigned long long modpow(unsigned long long a, unsigned long long b, unsigned long long m) {
    unsigned long long res = 1;
    a %= m;
    while (b) {
        if (b & 1) res = (unsigned __int128)res * a % m;
        a = (unsigned __int128)a * a % m;
        b >>= 1;
    }
    return res;
}
```

```
/* Returns true if n is prime, false otherwise. */
bool prime(unsigned long long n) {
    if (n < 3 || n % 2 == 0) return n == 2;
    if (n == 3 || n == 5 || n == 13 || n == 19 || n == 73 || n == 193 || n == 407521 || n == 299210837)
        return true;                                                                    /// two
    int s = __countr_zero(n - 1), pr[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};  /// two
    // int s = __countr_zero(n - 1), pr[] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37 }; // one
    unsigned long long d = (n - 1) >> s, x, y;
    for (int p : pr) {
        // if (p >= n) break; // one
        x = modpow(p, d, n);
        for (int j = 0; j < s; ++j) {
            y = (unsigned __int128)x * x % n;
            if (y == 1 && x != 1 && x != n - 1) return false;
            x = y;
        }
        if (y != 1) return false;
    }
    return true;
}
```

```
/* Computes non-trivial factor of n. */
unsigned long long pollards_rho(unsigned long long n) {
    if (n % 2 == 0) return 2;
    mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
    while (true) {
        unsigned long long c = rng() % (n - 2) + 1, x = rng() % n, y = x, d = 1;
        while (d == 1) {
            x = ((__int128)x * x % n + c) % n;
            y = ((__int128)y * y % n + c) % n;
            y = ((__int128)y * y % n + c) % n;
            d = gcd(x > y ? x - y : y - x, n);
        }
        if (d != n) return d;
    }
}
```

```
/* Computes factors of n. */
void factorize(unsigned long long n, map<unsigned long long, int>& fs) {
```

```
    if (n == 1) return;
    if (prime(n)) {
        ++fs[n];
        return;
    }
    unsigned long long f = pollards_rho(n);
    factorize(f, fs);
    factorize(n / f, fs);
}
```

```
/* Double can be replaced with int for INT_SAFE functions.
In that case EPS comparisons can be ignonred. */
typedef double coord;
struct Point {
    coord x, y;
    Point() : x(0), y(0) {}
    Point(coord X, coord Y) : x(X), y(Y) {}
    bool operator==(const Point& p) {
        const double EPS = 1E-9;
        return abs(x - p.x) < EPS && abs(y - p.y) < EPS;
    }
    bool operator<(const Point& p) {
        if (x < p.x) return true;
        if (x == p.x && y < p.y) return true;
        return false;
    }
    Point operator+(const Point& p) {
        return Point(x + p.x, y + p.y);
    }
    Point operator-(const Point& p) {
        return Point(x - p.x, y - p.y);
    }
    Point operator*(const coord c) {
        return Point(c * x, c * y);
    }
    Point operator/(const coord c) {
        return Point(x / c, y / c);
    }
    /* Inner product. Equiv to norm(this)*norm(p)*cos(angle(this,p)). */
    coord operator*(const Point& p) {
        return x * p.x + y * p.y;
    }
};
```

```
/* Computes norm of vector v. */
coord vec_norm(Point v) {
    return sqrt(v * v);
}
```

```
/* Rotates vector v by phi radians. */
Point rotate(Point v, double phi) {
    double sn = sin(phi), cs = cos(phi);
    return Point(v.x * cs - v.y * sn, v.x * sn + v.y * cs);
}
```

```
/* INT_SAFE Computes vector orthogonal to v. */
Point orth(Point v) {
    return Point(v.y, -v.x);
}
```

```
/* INT_SAFE Computes angle between u and v. */
double angle(Point u, Point v) {
    return acos(static_cast<double>(u * v) / (vec_norm(u) * vec_norm(v)));
}
```

```
/* INT_SAFE Computes the cross product of a and b. Equiv to vec_norm(a)*norm(b)*sin(angle(u,v)). */
coord cross_product(Point a, Point b) {
    return a.x * b.y - a.y * b.x;
}
```

```
/* INT_SAFE Returns 1 if p lies counter clockwise line, -1 one if it lies clockwise, and 0 otherwise. */
int ccw(Point a0, Point a1, Point p) {
    const double EPS = 1E-9;
    coord d1         = (a1.x - a0.x) * (p.y - a0.y);
    coord d2         = (p.x - a0.x) * (a1.y - a0.y);
    return (d1 - d2 > EPS) - (d2 - d1 > EPS);
```

```
}
```

```
/* Computes projection of p onto line(segment) a0 a1. */
Point closest_point(Point a0, Point a1, Point p) {
    /* For segment instead of line. */
    // if ((a0 - a1) * (p - a0) > 0) return a0;
    // if ((a1 - a0) * (p - a1) > 0) return a1;
    Point d = a1 - a0;
    return a0 + d * (d * (p - a0)) / (d * d);
}
```

```
/* INT_SAFE Returns 0 if p does not lie on segment, 1 if p strictly lies on segment and 2 if p lies on end point. */
int on_segment(Point a0, Point a1, Point p) {
    const double EPS = 1E-9;
    if (ccw(a0, a1, p)) return 0;
    coord cx = (p.x - a0.x) * (p.x - a1.x);
    coord cy = (p.y - a0.y) * (p.y - a1.y);
    if (cx > EPS || cy > EPS) return 0;
    if (cx < -EPS || cy < -EPS) return 1;
    return 2;
}
```

```
/* INT_SAFE Returns 0 if segments do not intersect, 1 if they strictly intersect and 2 if they intersect on end point.
 */
int segment_intersect(Point a0, Point a1, Point b0, Point b1) {
    int c1 = ccw(a0, a1, b0), c2 = ccw(a0, a1, b1);
    int c3 = ccw(b0, b1, a0), c4 = ccw(b0, b1, a1);
    if (c1 * c2 > 0 || c3 * c4 > 0) return 0;
    if ((c1 | c2 | c3 | c4) == 0) {  // Two segments lie on the same line.
        c1 = on_segment(a0, a1, b0);
        c2 = on_segment(a0, a1, b1);
        c3 = on_segment(b0, b1, a0);
        c4 = on_segment(b0, b1, a1);
        if (c1 && c2 && c3 && c4) return 2;
        if (!c1 && !c2 && !c3 && !c4) return 0;
        return max({c1, c2, c3, c4});
    }
    return (c1 && c2 && c3 && c4) ? 1 : 2;
}
```

```
/* Returns (INF, 0) if the lines are the same, (INF, _) if they are parallel and intersection point otherwise. */
Point line_intersect(Point a0, Point a1, Point b0, Point b1) {
    const double EPS = 1E-9;
    Point d13 = a0 - b0, d43 = b1 - b0, d21 = a1 - a0;
    double un = cross_product(d43, d13), ud = cross_product(d21, d43);
    if (abs(ud) < EPS) return Point(numeric_limits<double>::infinity(), un);
    return a0 + d21 * un / ud;
}
```

```
/* INT_SAFE Computes twice the area of poly. */
coord poly_2area(vector<Point>& poly) {
    int n      = poly.size();
    coord area = 0;
    for (int i = 0; i < n; ++i) area += (poly[(i + 1) % n].x - poly[i].x) * (poly[(i + 1) % n].y + poly[i].y);
    return abs(area);
}
```

```
/* INT_SAFE Returns 0 if p lies outside poly, 1 if p lies strictly inside poly and 2 if p lies on an edge of poly. */
int in_convex_poly(vector<Point>& poly, Point p) {
    const double EPS = 1E-9;
    int n            = poly.size();
    coord area2 = poly_2area(poly), parea2 = 0;
    for (int i = 0; i < n; ++i) {
        if (on_segment(poly[i], poly[(i + 1) % n], p)) return 2;
        parea2 += abs(cross_product(poly[i] - p, poly[(i + 1) % n] - p));
    }
    return abs(area2 - parea2) < EPS;
}
```

```
/* INT_SAFE Returns 0 if p lies outside poly, 1 if p lies strictly inside poly and 2 if p lies on an edge of poly. */
int in_concave_poly(vector<Point>& poly, Point p) {
    const double EPS = 1E-9;
    int intersects = 0, n = poly.size();
    for (int i = 0; i < n; ++i) {
        Point a = poly[i], b = poly[(i + 1) % n];
        if (on_segment(a, b, p)) return 2;
```

```
        coord diff1 = a.y - p.y, diff2 = b.y - p.y;
        if ((diff1 <= EPS && diff2 <= EPS) || (diff1 > EPS && diff2 > EPS)) continue;
        Point c = p - a, d = b - a;
        if (d.y < 0) {
            if (c.x * d.y > c.y * d.x) ++intersects;
        } else if (c.x * d.y < c.y * d.x) {
            ++intersects;
        }
    }
    return intersects & 1;
}
```

```
/* INT_SAFE Returns smallest convex hull of points given in counter clockwise order. */
vector<Point> convex_hull(vector<Point> points) {
    points.resize(distance(points.begin(), unique(points.begin(), points.end())));
    sort(points.begin(), points.end());
    int n = points.size(), k = 0;
    if (n <= 1) return points;
    vector<Point> hull(2 * n);
    for (int i = 0; i < n; ++i) {
        while (k > 1 && ccw(hull[k - 2], hull[k - 1], points[i]) <= 0) --k;
        hull[k++] = points[i];
    }
    int t = k;
    for (int i = n - 1; i >= 0; --i) {
        while (k > t && ccw(hull[k - 2], hull[k - 1], points[i]) <= 0) --k;
        hull[k++] = points[i];
    }
    hull.resize(k > 1 ? k - 1 : k);
    return hull;
}
```

```
/* Reads a decimal integer from input and returns it as __int128. */
__int128 i128in() {
    string s;
    cin >> s;
    __int128 res = 0;
    bool neg     = s[0] == '-';
    int i = neg, n = s.size();
    for (; i < n; ++i) res = res * 10 + (s[i] - '0');
    return neg ? -res : res;
}
```

```
/* Prints a __int128 integer to standard output. */
void i128out(__int128 x) {
    if (!x) {
        cout << '0';
        return;
    }
    if (x < 0) {
        cout << '-';
        x = -x;
    }
    char buf[50] = {0};
    int i        = 0;
    while (x > 0) {
        buf[i++] = '0' + (x % 10);
        x /= 10;
    }
    reverse(buf, buf + i);
    cout << buf;
}
```