Template

```cpp
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef pair<ll, ll> P;

#define each(i,a) for (auto&& i : a)
#define FOR(i,a,b) for (ll i=(a),__last_##i=(b);i<__last_##i;i++)
#define RFOR(i,a,b) for (ll i=(b)-1,__last_##i=(a);i>=__last_##i;i--)
#define REP(i,n) FOR(i,0,n)
#define RREP(i,n) RFOR(i,0,n)
#define __GET_MACRO3(_1, _2, _3, NAME, ...) NAME
#define rep(...) __GET_MACRO3(__VA_ARGS__, FOR, REP)(__VA_ARGS__)
#define rrep(...) __GET_MACRO3(__VA_ARGS__, RFOR, RREP)(__VA_ARGS__)
#define pb push_back
#define all(a) (a).begin(),(a).end()
#define chmin(x,v) x = min(x, v)
#define chmax(x,v) x = max(x, v)

const ll linf = 1e18;
const double eps = 1e-12;
const double pi = acos(-1);

template<typename T>
istream& operator>>(istream& is, vector<T>& vec) {
    each(x,vec) is >> x;
    return is;
}
template<typename T>
ostream& operator<<(ostream& os, const vector<T>& vec) {
    rep(i,vec.size()) {
        if (i) os << " ";
        os << vec[i];
    }
    return os;
}
template<typename T>
ostream& operator<<(ostream& os, const vector< vector<T> >& vec) {
    rep(i,vec.size()) {
        if (i) os << endl;
        os << vec[i];
    }
    return os;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
}
```

Ahocorasick

```cpp
class AhoCorasick {
    void clear_graph() {
        root.child.clear();
        root.pattern.clear();
    }
    void generate_trie(const vector<string>& patterns) {
        ll n = patterns.size();
        rep(i, n) {
            Node* t = &root;
            each(c, patterns[i]) {
                t = &(t->child[c]);
            }
            t->pattern.push_back(i);
        }
    }
    void add_failure_edge() {
        queue<Node*> Q; Q.push(&root);
        //幅優先探索で帰納的に失敗時の遷移辺を追加していく
        while (!Q.empty()) {
            Node* t = Q.front(); Q.pop();
            each(p, t->child) {
                Q.push(&(p.second));
                char c = p.first;
                Node* node = &(p.second); // 文字 c で遷移する頂点
                Node* anode = t->failure; // 失敗したときの遷移先
                while ( anode != NULL && anode->child.count(c) == 0 ) {
                    anode = anode->failure;
                }
                //遷移失敗時に続けられる別の頂点へ遷移
                if (anode == NULL) {
                    node->failure = &root;
```

```
                }
                else {
                    node->failure = &(anode->child[c]);
                }
                //マッチするパターンを追加
                each(to, node->failure->pattern) {
                    node->pattern.pb(to);
                }
                //メモリ食いすぎ回避のため切り詰めておく
                vector<size_t>(node->pattern).swap(node->pattern);
            }
        }
    }
    //Pattern Match Automaton を構築
    void make_PMA(const vector<string>& patterns) {
        clear_graph();
        generate_trie(patterns);
        add_failure_edge();
    }
public:
    struct Node {
        map<char,Node> child; //遷移辺 (!)
        vector<size_t> pattern; //マッチするパターン (の index)
        Node* failure; //遷移失敗時の遷移先ノード
        Node():failure(NULL) {}
    };
    Node root;
    AhoCorasick(const vector<string>& patterns) {
        make_PMA(patterns);
    }
    pair<Node*, vector<ll>> find(Node* node, char c) {
        vector<ll> res;
        while (node != NULL && node->child.count(c) == 0) {
            node = node->failure;
        }
        if (node == NULL) node = &root;
        else node = &(node->child[c]);
        each(ptn, node->pattern) {
            res.pb(ptn);
        }
        return pair<Node*, vector<ll>>(node, res);
    }
};
```

Convex Hull

```
// 傾き単調減少
// get クエリ単調増加
class ConvexHull {
    deque<ll> a, b;
    bool check(ll f1, ll f2, ll aa, ll bb) {
        return (a[f2] - a[f1]) * (bb - b[f2]) >= (b[f2] - b[f1]) * (aa - a[f2]);
    }
    ll f(ll fid, ll x) {
        return a[fid] * x + b[fid];
    }
public:
    void add(ll aa, ll bb) {
        while (a.size() >= 2 && check(a.size()-2, a.size()-1, aa, bb)) {
            a.pop_back();
            b.pop_back();
        }
        a.push_back(aa);
        b.push_back(bb);
    }
    ll get_min(ll x) {
        while (a.size() >= 2 && f(0, x) >= f(1, x)) {
            a.pop_front();
            b.pop_front();
        }
        return a[0] * x + b[0];
    }
};
```

FFT

```
typedef complex<double> C;
void dft(vector<C>& f, int s, int d, int n) {
    if (n == 1) return;
    dft(f, s, d*2, n/2);
    dft(f, s+d, d*2, n/2);
    vector<C> f0(n/2);
    vector<C> f1(n/2);
    for (int i = 0; i < f0.size(); ++i) f0[i] = f[s+2*i*d];
    for (int i = 0; i < f1.size(); ++i) f1[i] = f[s+(2*i+1)*d];
```

```
    C zeta(cos(2.0*pi/n), sin(2.0*pi/n));
    C z = 1;
    REP(i, n) {
        f[s+i*d] = f0[i % (n/2)] + z * f1[i % (n/2)];
        z *= zeta;
    }
}
void idft(vector<C>& f, int s, int d, int n) {
    if (n == 1) return;
    idft(f, s, d*2, n/2);
    idft(f, s+d, d*2, n/2);
    vector<C> f0(n/2);
    vector<C> f1(n/2);
    REP(i, f0.size()) f0[i] = f[s+2*i*d];
    REP(i, f1.size()) f1[i] = f[s+(2*i+1)*d];
    C zeta(cos(2.0*pi/n), -sin(2.0*pi/n));
    C z = C(1, 0);
    REP(i, n) {
        f[s+i*d] = f0[i % (n/2)] + z * f1[i % (n/2)];
        z *= zeta;
    }
}
int pow_2_at_least(int th) {
    int ret = 1;
    while (ret <= th) ret <<= 1;
    return ret;
}
void dft(vector<C>& f) {
    int n = pow_2_at_least(f.size() - 1);
    while (f.size() < n) f.push_back( C(0,0) );
    dft(f, 0, 1, n);
}
void idft(vector<C>& f) {
    int n = pow_2_at_least(f.size() - 1);
    while (f.size() < n) f.push_back( C(0,0) );
    idft(f, 0, 1, n);
}
vector<C> multiply(vector<C> g, vector<C> h) {
    int n = pow_2_at_least(g.size() + h.size() - 1);
    while (g.size() < n) g.push_back( C(0,0) );
    while (h.size() < n) h.push_back( C(0,0) );
    dft(g);
    dft(h);

    vector<C> f(n);
    for (int i = 0; i < n; ++i) {
        f[i] = g[i] * h[i];
    }
    idft(f);

    vector<C> ret(n);
    for (int i = 0; i < n; ++i) {
        ret[i] = f[i]/C(n,0);
    }
    return ret;
}
```

Geometry

```
/* 幾何の基本 */
typedef long double ld;
typedef complex<ld> Point;
namespace std {
    bool operator<(const Point &lhs, const Point &rhs) {
        if (lhs.real() < rhs.real() - eps) return true;
        if (lhs.real() > rhs.real() + eps) return false;
        return lhs.imag() < rhs.imag();
    }
}
// 点の入力
Point input_point() {
    ld x, y;
    cin >> x >> y;
    return Point(x, y);
}
// 誤差つき等号判定
bool eq(ld a, ld b) {
    return (abs(a - b) < eps);
}
// 内積
ld dot(Point a, Point b) {
    return real(conj(a) * b);
}
// 外積
ld cross(Point a, Point b) {
```

```
    return imag(conj(a) * b);
}
// 直線の定義
class Line {
public:
    Point a, b;
    Line () : a(Point(0, 0)), b(Point(0, 0)) {}
    Line (Point a, Point b) : a(a), b(b) {}
};
// 円の定義
class Circle {
public:
    Point p;
    ld r;
    Circle () : p(Point(0, 0)), r(0) {}
    Circle (Point p, ld r) : p(p), r(r) {}
};
// CCW
int ccw (Point a, Point b, Point c) {
    b -= a; c -= a;
    if (cross(b, c) > eps) return 1; // a,b,c が反時計周りの順に並ぶ
    if (cross(b, c) < -eps) return -1; // a,b,c が時計周りの順に並ぶ
    if (dot(b, c) < 0) return 2; // c,a,b の順に直線に並ぶ
    if (norm(b) < norm(c)) return -2; // a,b,c の順に直線に並ぶ
    return 0; // a,c,b の順に直線に並ぶ
}


/* 交差判定 */

// 直線と直線の交差判定
bool isis_ll (Line l, Line m) {
    return !eq(cross(l.b - l.a, m.b - m.a), 0);
}
// 直線と線分の交差判定
bool isis_ls (Line l, Line s) {
    return isis_ll(l, s) &&
        (cross(l.b - l.a, s.a - l.a) * cross(l.b - l.a, s.b - l.a) < eps);
}
// 線分と線分の交差判定
bool isis_ss(Line s, Line t) {
    return ccw(s.a, s.b, t.a) * ccw(s.a, s.b, t.b) <= 0 &&
        ccw(t.a, t.b, s.a) * ccw(t.a, t.b, s.b) <= 0;
}
// 点の直線上判定
bool isis_lp (Line l, Point p) {
    return (abs(cross(l.b - p, l.a - p)) < eps);
}
// 点の線分上判定
bool isis_sp (Line s, Point p) {
    return (abs(s.a - p) + abs(s.b - p) - abs(s.b - s.a) < eps);
}
// 垂線の足
Point proj (Line l, Point p) {
    ld t = dot(p - l.a, l.a - l.b) / norm(l.a - l.b);
    return l.a + t * (l.a - l.b);
}
// 直線と直線の交点
Point is_ll (Line s, Line t) {
    Point sv = s.b - s.a, tv = t.b - t.a;
    assert(cross(sv, tv) != 0);
    return s.a + sv * cross(tv, t.a - s.a) / cross(tv, sv);
}
// 直線と点の距離
ld dist_lp (Line l, Point p) {
    return abs(p - proj(l, p));
}
// 直線と直線の距離
ld dist_ll (Line l, Line m) {
    return isis_ll(l, m) ? 0 : dist_lp(l, m.a);
}
// 直線と線分の距離
ld dist_ls (Line l, Line s) {
    return isis_ls(l, s) ? 0 : min(dist_lp(l, s.a), dist_lp(l, s.b));
}
// 線分と点の距離
ld dist_sp (Line s, Point p) {
    Point r = proj(s, p);
    return isis_sp(s, r) ? abs(r - p) : min(abs(s.a - p), abs(s.b - p));
}
// 線分と線分の距離
ld dist_ss (Line s, Line t) {
    if (isis_ss(s, t)) return 0;
```

```
    return min({dist_sp(s, t.a), dist_sp(s, t.b), dist_sp(t, s.a), dist_sp(t, s.b)});
}

/* 円 */

// 円と円の交点
vector<Point> is_cc (Circle c1, Circle c2){
    vector<Point> res;
    ld d = abs(c1.p - c2.p);
    ld rc = (d * d + c1.r * c1.r - c2.r * c2.r) / (2 * d);
    ld dfr = c1.r * c1.r - rc * rc;
    if (abs(dfr) < eps) dfr = 0.0;
    else if (dfr < 0.0) return res; // no intersection
    ld rs = sqrt(dfr);
    Point diff = (c2.p - c1.p) / d;
    res.push_back(c1.p + diff * Point(rc, rs));
    if (dfr != 0.0) res.push_back(c1.p + diff * Point(rc, -rs));
    return res;
}
// 円と直線の交点
vector<Point> is_lc (Circle c, Line l){
    vector<Point> res;
    ld d = dist_lp(l, c.p);
    if (d < c.r + eps){
        ld len = (d > c.r) ? 0.0 : sqrt(c.r * c.r - d * d); //safety;
        Point nor = (l.a - l.b) / abs(l.a - l.b);
        res.push_back(proj(l, c.p) + len * nor);
        res.push_back(proj(l, c.p) - len * nor);
    }
    return res;
}
// 円と線分の交点
vector<Point> is_sc(Circle c, Line l){
    vector<Point> v = is_lc(c, l), res;
    for (Point p : v)
        if (isis_sp(l, p)) res.push_back(p);
    return res;
}
// 円と点の接線
vector<Line> tangent_cp(Circle c, Point p) {
    vector<Line> ret;
    Point v = c.p - p;
    ld d = abs(v);
    ld l = sqrt(norm(v) - c.r * c.r);
    if (isnan(l)) { return ret; }
    Point v1 = v * Point(l / d,        c.r / d);
    Point v2 = v * Point(l / d, -c.r / d);
    ret.push_back(Line(p, p + v1));
    if (l < eps) return ret;
    ret.push_back(Line(p, p + v2));
    return ret;
}
// 円と円の接線
vector<Line> tangent_cc(Circle c1, Circle c2) {
    vector<Line> ret;
    if (abs(c1.p - c2.p) - (c1.r + c2.r) > -eps) {
        Point center = (c1.p * c2.r + c2.p * c1.r) / (c1.r + c2.r);
        ret = tangent_cp(c1, center);
    }
    if (abs(c1.r - c2.r) > eps) {
        Point out = (-c1.p * c2.r + c2.p * c1.r) / (c1.r - c2.r);
        vector<Line> nret = tangent_cp(c1, out);
        ret.insert(ret.end(), ALL(nret));
    }
    else {
        Point v = c2.p - c1.p;
        v /= abs(v);
        Point q1 = c1.p + v * Point(0,        1) * c1.r;
        Point q2 = c1.p + v * Point(0, -1) * c1.r;
        ret.push_back(Line(q1, q1 + v));
        ret.push_back(Line(q2, q2 + v));
    }
    return ret;
}


/* 多角形 */
typedef vector<Point> Polygon;
// 面積
ld area(const Polygon &p) {
    ld res = 0;
    int n = p.size();
    REP(j,n) res += cross(p[j], p[(j+1)%n]);
```

```
    return res / 2;
}
// 多角形の回転方向
bool is_counter_clockwise (const Polygon &poly) {
    ld angle = 0;
    int n = poly.size();
    REP(i,n) {
        Point a = poly[i], b = poly[(i+1)%n], c = poly[(i+2)%n];
        angle += arg((c - b) / (b - a));
    }
    return angle > eps;
}
// 円の内外判定
//  0 => out
//       1 => on
//       2 => in
int is_in_polygon (const Polygon &poly, Point p) {
    ld angle = 0;
    int n = poly.size();
    REP(i,n) {
        Point a = poly[i], b = poly[(i+1)%n];
        if (isis_sp(Line(a, b), p)) return 1;
        angle += arg((b - p) / (a - p));
    }
    return eq(angle, 0) ? 0 : 2;
}
// 凸包
Polygon convex_hull(vector<Point> ps) {
    int n = ps.size();
    int k = 0;
    sort(ps.begin(), ps.end());
    Polygon ch(2 * n);
    for (int i = 0; i < n; ch[k++] = ps[i++])
        while (k >= 2 && ccw(ch[k - 2], ch[k - 1], ps[i]) <= 0) --k;
    for (int i = n - 2, t = k + 1; i >= 0; ch[k++] = ps[i--])
        while (k >= t && ccw(ch[k - 2], ch[k - 1], ps[i]) <= 0) --k;
    ch.resize(k - 1);
    return ch;
}
// 凸カット
Polygon convex_cut(const Polygon &ps, Line l) {
    int n = ps.size();
    Polygon Q;
    REP(i,n) {
        Point A = ps[i], B = ps[(i+1)%n];
        Line m = Line(A, B);
        if (ccw(l.a, l.b, A) != -1) Q.push_back(A);
        if (ccw(l.a, l.b, A) * ccw(l.a, l.b, B) < 0 && isis_ll(l, m))
            Q.push_back(is_ll(l, m));
    }
    return Q;
}


/* アレンジメント */
void add_point(vector<Point> &ps, Point p) {
    for (Point q : ps) if (abs(q - p) < eps) return;
    ps.push_back(p);
}

typedef int Weight;

struct Edge { int from, to; Weight weight; };

typedef vector<Edge> Edges;
typedef vector<Edges> Graph;

void add_edge(Graph &g, int from, int to, Weight weight) {
    g[from].push_back((Edge){from, to, weight});
}
Graph segment_arrangement(const vector<Line> &s, const vector<Point> &p) {
    int n = p.size(), m = s.size();
    Graph g(n);
    REP(i,m) {
        vector<pair<ld,int>> vec;
        REP(j,n) if (isis_sp(s[i], p[j]))
            vec.emplace_back(abs(s[i].a - p[j]), j);
        sort(ALL(vec));
        REP(j,vec.size()-1) {
            int from = vec[j].second, to = vec[j+1].second;
            add_edge(g, from, to, abs(p[from] - p[to]));
        }
    }
```

```
    return g;
}
Graph circle_arrangement(const vector<Circle> &c, const vector<Point> &p) {
    int n = p.size(), m = c.size();
    Graph g(n);
    REP(i,m) {
        vector<pair<ld,int>> vec;
        REP(j,n) if (abs(abs(c[i].p - p[j]) - c[i].r) < eps)
            vec.emplace_back(arg(c[i].p - p[j]), j);
        sort(ALL(vec));
        REP(j,vec.size()-1) {
            int from = vec[j].second, to = vec[j+1].second;
            ld angle = vec[j+1].first - vec[j].first;
            add_edge(g, from, to, angle * c[i].r);
        }
        if (vec.size() >= 2) {
            int from = vec.back().second, to = vec.front().first;
            ld angle = vec.front().first - vec.back().first;
            add_edge(g, from, to, angle * c[i].r);
        }
    }
    return g;
}


/* 双対グラフ */

// 線分集合は既にアレンジメントされていなければならない.
// 内側の円は時計回りで，外側の円は反時計回りで得られる.
// 変数 polygon は，vector<int> で表される多角形の集合であり，
// vector<int> で表される 多角形の i 番目は，その頂点の頂点集合 p における番号である.
vector<vector<int>> polygon;
vector<int> seg2p[1024][1024];

Graph dual_graph(const vector<Line> &s, const vector<Point> &p) {
    int N = p.size();
    polygon.clear();
    REP(i,1024) REP(j,1024) seg2p[i][j].clear();
    vector<vector<tuple<ld,int,bool>>> tup(N);
    REP(i,s.size()) {
        int a = -1, b = -1;
        REP(j,N) if (abs(s[i].a - p[j]) < eps) a = j;
        REP(j,N) if (abs(s[i].b - p[j]) < eps) b = j;
        assert(a >= 0 && b >= 0);
        tup[a].emplace_back(arg(s[i].b - s[i].a), b, false);
        tup[b].emplace_back(arg(s[i].a - s[i].b), a, false);
    }
    REP(i,N) sort(ALL(tup[i]));
    REP(i,N) {
        REP(j,tup[i].size()) {
            ld angle; int pos = j, from = i, to; bool flag;
            tie(angle, to, flag) = tup[i][j];
            if (flag) continue;
            vector<int> ps;
            while (!flag) {
                ps.push_back(from);
                get<2>(tup[from][pos]) = true;
                seg2p[from][to].push_back(polygon.size());
                seg2p[to][from].push_back(polygon.size());
                angle += pi + eps;
                if (angle > pi) angle -= 2 * pi;
                auto it = lower_bound(ALL(tup[to]), make_tuple(angle, 0, false));
                if (it == tup[to].end()) it = tup[to].begin();
                from = to; tie(angle, to, flag) = *it;
                pos = it - tup[from].begin();
            }
            polygon.push_back(ps);
        }
    }
    Graph g(polygon.size());
    REP(i,N) REP(j,i) {
        if (seg2p[i][j].size() == 2) {
            int from = seg2p[i][j][0], to = seg2p[i][j][1];
            g[from].push_back((Edge){from, to});
            g[to].push_back((Edge){to, from});
        }
    }
    return g;
}


```

```
/* ビジュアライザ */
const ld zoom = 25;
const ld centerX = 6;
const ld centerY = 5;

void change_color(int r, int g, int b) {
    fprintf(stderr, "c.strokeStyle = 'rgb(%d, %d, %d)';\n", r, g, b);
}

int cordx(Point p) { return 400 + zoom * (p.real() - centerX); }
int cordy(Point p) { return 400 - zoom * (p.imag() - centerY); }

#define cord(p) cordx(p),cordy(p)

void draw_point(Point p) {
    fprintf(stderr, "circle(%d, %d, %d)\n", cord(p), 2);
}

void draw_segment(Line l) {
    fprintf(stderr, "line(%d, %d, %d, %d)\n", cord(l.a), cord(l.b));
}

void draw_line(Line l) {
    Point v = l.b - l.a;
    Line m(l.a - v * Point(1e4, 0), l.b + v * Point(1e4, 0));
    fprintf(stderr, "line(%d, %d, %d, %d)\n", cord(m.a), cord(m.b));
}

void draw_polygon(const Polygon &p) {
    int n = p.size();
    REP(i,n) draw_segment(Line(p[i], p[(i+1)%n]));
}

void draw_circle(Circle c) {
    fprintf(stderr, "circle(%d, %d, %d)\n", cord(c.p), (int)(zoom * c.r));
}
```

KD

```
//shared_ptr使っているので低速
struct po {
    int index;
    vector<int> coors;
    po(int _d):coors(_d) {
        index = -1;
    }
    po() {}
};
template<class T>
class axisSorter {
    int k;
public:
    axisSorter(int _k) : k(_k) {}
    bool operator()(const T &a, const T &b) {
        return a.coors[k] < b.coors[k];
    }
};
long long int getdis(const po&l, const po&r) {
    long long int dis = 0;
    for (int i = 0; i < l.coors.size(); ++i) {
        dis += (l.coors[i] - r.coors[i])*(l.coors[i] - r.coors[i]);
    }
    return dis;
}
template<class T, int Dim = 2>
struct kdtree {
public:
    T val;
  shared_ptr<kdtree<T>> ltree, rtree;
    int depth;
    int axis;
    kdtree(const T &p_) :val(p_), ltree(nullptr), rtree(nullptr) {

    }

    kdtree(vector<T>&ps_, const int& l, const int& r, const int depth_ = 0) : ltree(nullptr), rtree(nullptr)
        init(ps_, l, r);
    }
~kdtree() {
    if (ltree != nullptr)        delete(ltree);

    if (rtree != nullptr)delete(rtree);
}
//直方体内にある点の数を求める。
    vector<T>query(const T & amin, const T&amax) {
        vector<T>ans;
        bool aok = true;
        for (int i = 0; i < Dim; ++i) {
```

```
            if (amin.coors[i] <= val.coors[i] && val.coors[i] <= amax.coors[i]) {

            }
            else {
                aok = false;
                break;
            }
        }
        if (aok) {
            ans.emplace_back(val);
        }
        axisSorter<T> as(axis);
        if (as(val, amax) || val.coors[axis] == amax.coors[axis]) {
            if (rtree != nullptr) {
                vector<T>tans(rtree->query(amin, amax));
                ans.insert(ans.end(), tans.begin(), tans.end());
            }
        }
        if (as(amin, val) || val.coors[axis] == amin.coors[axis]) {
            if (ltree != nullptr) {
                vector<T>tans(ltree->query(amin, amax));
                ans.insert(ans.end(), tans.begin(), tans.end());
            }
        }
        return ans;
    }
//最近傍点を求める。
void get_closest(const T& apo, long long int &ans) {
    ans = min(ans, getdis(apo, val));
    axisSorter<T> as(axis);
    if (as(apo, val) || val.coors[axis] == apo.coors[axis]) {
        if (ltree)ltree->get_closest(apo, ans);
        long long int dis = apo.coors[axis] - val.coors[axis];
        if (dis*dis >= ans)return;
        else {
            if (rtree)rtree->get_closest(apo, ans);
        }
    }
    else {
        if (rtree)rtree->get_closest(apo, ans);
        long long int dis = val.coors[axis] - apo.coors[axis];
        if (dis*dis >= ans)return;
        else {
            if (ltree)ltree->get_closest(apo, ans);
        }
    }
}
private:
    void init(vector<T>&ps, const int& l, const int& r) {
        if (l >= r) {
            return;
        }
        const int mid = (l + r) / 2;
        nth_element(ps.begin() + l, ps.begin() + mid, ps.begin() + r, axisSorter<T>(axis));
        val = ps[mid];
        ltree = make_kdtree(ps, l, mid, depth + 1);
        rtree = make_kdtree(ps, mid + 1, r, depth + 1);
    }

};
//[l..r)
template<class T>
unique_ptr<kdtree<T>>make_kdtree(vector<T>&ps_, const int& l, const int& r, const int& depth = 0) {
    if (l >= r)return nullptr;
    else {
        return make_unique<kdtree<T>>(ps_, l, r, depth);
    }
}
```

Math

```
const ll mod = ;
ll mul(ll a, ll b) {
    return a * b % mod;
}
ll mul(initializer_list<ll> t) {
    ll res = 1;
    each(v, t) res = mul(res, v);
    return res;
}
ll add(ll a, ll b) {
    return (a + b) % mod;
}
ll add(initializer_list<ll> t) {
    ll res = 0;
```

```
    each(v, t) res = add(res, v);
    return res;
}
ll sub(ll a, ll b) {
    return (a - b + mod) % mod;
}
ll sub(initializer_list<ll> t) {
    auto it = t.begin();
    ll res = *(it++);
    while (it != t.end()) {
        res = sub(res, *(it++));
    }
    return res;
}
ll power(ll x, ll n) {
    ll res = 1;
    for (ll i = 1; i <= n; i <<= 1) {
        if (i & n) res = mul(res, x);
        x = mul(x, x);
    }
    return res;
}
ll inv(ll n) {
    return power(n, mod-2);
}
ll divi(ll a, ll b) {
    return mul(a, inv(b));
}
ll divi(initializer_list<ll> t) {
    auto it = t.begin();
    ll res = *(it++);
    while (it != t.end()) {
        res = divi(res, *(it++));
    }
    return res;
}
vector<ll> fact;
void init_fact(ll n) {
    fact.assign(n+1, 1);
    FOR(i, 1, fact.size()) {
        fact[i] = mul(fact[i-1], i);
    }
}

ll comb(ll n, ll r) {
    if (r < 0) return 0;
    if (r > n) return 0;
    return divi(fact[n], mul(fact[r], fact[n-r]));
}
```

Max Flow (Dinic)

```
class MaxFlow {
public:
    struct Edge {
        ll to, cap, rev;
    };
    vector<vector<Edge>> G;
    vector<ll> iter;
private:
    bool is_debug;
    ll V;
    vector<ll> bfs(ll s) {
        vector<ll> dist(V, linf);
        dist[s] = 0;
        queue<ll> Q; Q.push(s);
        while ( !Q.empty() ) {
            ll v = Q.front(); Q.pop();
            each(e, G[v]) {
                if (e.cap > 0 && dist[e.to] == linf) {
                    dist[e.to] = dist[v]+1;
                    Q.push(e.to);
                }
            }
        }
        return dist;
    }
    ll dfs(ll v, ll t, ll f, const vector<ll>& dist, vector<bool>& used) {
        if (v == t) return f;
        if (used[v]) return 0;
        used[v] = true;
        for (ll& i = iter[v]; i < G[v].size(); ++i) {
            Edge& e = G[v][i];
            if (e.cap > 0 && dist[e.to] == dist[v]+1) {
                ll d = dfs(e.to, t, min(f, e.cap), dist, used);
                if (d > 0) {
```

```
                    e.cap -= d;
                    G[e.to][e.rev].cap += d;
                    return d;
                }
            }
        }
        return 0;
    }
public:
    const vector<vector<Edge>> Graph() {
        return G;
    }
    MaxFlow(ll V, bool is_debug=false) : V(V), G(V), is_debug(is_debug) {}
    void init(ll n) {
        V = n;
        G.assign(V, vector<Edge>());
    }
    void add(ll from, ll to, ll cap) {
        if (is_debug) cout << "ADD: " << from << " " << to << " " << cap << endl;
        assert(V > 0);
        G[from].pb({to, cap, (ll)G[to].size()});
        G[to].pb({from, 0, (ll)G[from].size()-1});
    }
    // S -> s, T -> t に inf は自力で
    void add(ll from, ll to, ll min_flow, ll cap, ll S, ll T) {
        if (is_debug) cout << endl << "ADD_MIN:" << from << " " << to << " " << min_flow << " " << cap << end
        add(from, to, cap-min_flow);
        add(S, to, min_flow);
        add(from, T, cap);
        if (is_debug) cout << endl;
    }
    ll flow(ll s, ll t, ll f=linf) {
        ll res = 0;
        while (f > 0) {
            vector<ll> dist = bfs(s);
            if (dist[t] == linf) break;
            iter.assign(G.size(), 0);
            while (f > 0) {
                vector<bool> used(V, false);
                ll df = dfs(s, t, f, dist, used);
                if (df == 0) break;
                f -= df;
                res += df;
            }
        }
        return res;
    }
};
```

Min Cost Flow

```
const ll maxV = 3e5;

struct Edge {
    ll to, cap, cost, rev;
};

vector< vector<Edge> > G;

void add_edge(ll from, ll to, ll cap, ll cost) {
    if (from < 0 || to < 0) return;
    G[from].push_back({to, cap, cost, (ll)G[to].size()});
    G[to].push_back({from, 0, -cost, (ll)G[from].size()-1});
}

ll dist[maxV], h[maxV] = {0}, prevV[maxV], prevE[maxV];
ll min_cost_flow(ll s, ll t, ll f, bool is_ford_first = false) {
    ll res = 0;
    while (f > 0) {
        fill(dist, dist+maxV, linf); dist[s] = 0;
        if (is_ford_first) {
            while (1) {
                bool is_update = false;
                rep(v, G.size()) {
                    if (dist[v] == linf) continue;
                    rep(i, G[v].size()) {
                        const Edge& e = G[v][i];
                        if (e.cap > 0 && dist[v] + e.cost < dist[e.to]) {
                            dist[e.to] = dist[v] + e.cost;
                            prevV[e.to] = v;
                            prevE[e.to] = i;
                            is_update = true;
                        }
                    }
                }
```

```
                if (!is_update) break;
            }
        }
        else {
            priority_queue<P, vector<P>, greater<P> > Q; Q.push({0, s});
            while ( !Q.empty() ) {
                P p = Q.top(); Q.pop();
                ll v = p.second;
                if (p.first > dist[v]) continue;
                for (ll i = 0; i < G[v].size(); ++i) {
                    Edge& e = G[v][i];
                    if (e.cap > 0 && dist[v]+e.cost+h[v]-h[e.to] < dist[e.to]) {
                        dist[e.to] = dist[v]+e.cost+h[v]-h[e.to];
                        prevV[e.to] = v;
                        prevE[e.to] = i;
                        Q.push({dist[e.to], e.to});
                    }
                }
            }
        }
        rep(i, G.size()) h[i] += dist[i];
        if (dist[t] == linf) {
            throw res;
        }
        ll d = f;
        for (ll v = t; v != s; v = prevV[v]) {
            d = min(d, G[prevV[v]][prevE[v]].cap);
        }
        f -= d;
        res += d * h[t];
        for (ll v = t; v != s; v = prevV[v]) {
            Edge& e = G[prevV[v]][prevE[v]];
            e.cap -= d;
            G[e.to][e.rev].cap += d;
        }
    }
    return res;
}
```

Potential Equation

```
// solve {x_i - x_j = c | (i, j, c) in E}
class Potential {
    vector<ll> par, h;
public:
    vector<ll> pot;
    Potential(ll size) : par(size, 0), h(size, 0), pot(size, 0) {
        rep(i, size) par[i] = i;
    }
    // u - v = cost?
    bool check(ll u, ll v, ll cost) {
        return pot[u] - pot[v] == cost;
    }
    // add: u - v = cost
    void add(ll u0, ll v0, ll cost) {
        ll u = root(u0), v = root(v0);
        if (u == v) {
            if (!check(u0, v0, cost)) throw -1;
            return;
        }
        if (h[u] > h[v]) {
            pot[v] = -cost + pot[u0] - pot[v0];
            par[v] = u;
        }
        else {
            pot[u] = cost + pot[v0] - pot[u0];
            par[u] = v;
        }
        if (h[u] == h[v]) ++h[u];
    }
    bool isUnited(ll u, ll v) {
        return root(u) == root(v);
    }
    ll root(ll v) {
        if (par[v] == v) return v;
        ll r = root(par[v]);
        pot[v] += pot[par[v]];
        return par[v] = r;
    }
};
```

Run Length Encoding

```
// ランレングス圧縮
vector<pair<char, int>> rle(string s) {
```

```
    char prev = '\0';
    int cnt = 0;
    vector<pair<char, int>> res;
    REP(i, s.size()+1) {
        if (i == s.size() || s[i] != prev) {
            if (prev != '\0') {
                res.pb({prev, cnt});
            }
            prev = s[i];
            cnt = 1;
        }
        else {
            ++cnt;
        }
    }
    return res;
}
```

Rolling Hash

```
ll mul(ll a, ll b, ll mod) {
    return a * b % mod;
}
ll add(ll a, ll b, ll mod) {
    return (a + b) % mod;
}
ll sub(ll a, ll b, ll mod) {
    return (a - b + mod) % mod;
}
ll power(ll x, ll n, ll mod) {
    ll res = 1;
    for (ll i = 1; i <= n; i <<= 1) {
        if (i & n) res = mul(res, x, mod);
        x = mul(x, x, mod);
    }
    return res;
}
ll inv(ll n, ll mod) {
    return power(n, mod-2, mod);
}

class RollingHash {
private:
    const ll A, mod;
    const string str;
    vector<ll> hash;
    vector<ll> make_hash(const string& s, const ll A, const ll mod) {
        vector<ll> res(s.size()+1, 0);
        ll coe = 1;
        REP(i, s.size()) {
            hash[i+1] = add(hash[i], mul(coe, s[i], mod), mod);
            coe = mul(coe, A, mod);
        }
        return res;
    }
public:
    RollingHash(const string& s, const ll A, const ll mod) : str(s), A(A), mod(mod), hash(make_hash(s, A, mod
    ll get(ll l, ll r) {
        return mul(sub(hash[r], hash[l], mod), inv(power(A, l, mod), mod), mod);
    }
    bool iseq(ll s1, ll s2, ll len) {
        return get(s1, s1+len) == get(s2, s2+len);
    }
};
```

Slide Maximum

```
class MaxSet {
    deque<P> deq;
public:
    MaxSet() {}
    ll get() {
        if (deq.size() == 0) return -linf;
        return deq.front().first;
    }
    // 値 x, 時刻 t
    ll add(ll x, ll t) {
        if (deq.size() > 0) assert(t > deq.back().second);
        while (deq.size() > 0 && deq.back().first <= x) {
            deq.pop_back();
        }
        deq.push_back(P(x, t));
    }
    // t 未満削除
    ll erase(ll t) {
        while (deq.size() > 0 && deq.front().second < t) {
```

```
            deq.pop_front();
        }
    }
};
```

Suffix Array

```
class SuffixArray {
    const ll n;
    const string str;
    vector<ll> sa, lcp;
public:
    SuffixArray(const string& s) : str(s), n(s.size()) {}
    vector<ll> make_sa() {
        sa.assign(n+1, 0);
        rep(i, n+1) sa[i] = i;
        vector<ll> rank(all(str));
        rank.pb(-1);
        auto f = [&](ll idx, ll len) {
            return idx + len <= n ? rank[idx+len] : -1;
        };
        for (ll k = 1; k <= n; k <<= 1) {
            auto compare = [&](ll a, ll b) {
                if (rank[a] != rank[b]) return rank[a] < rank[b];
                else return f(a, k) < f(b, k);
            };
            sort(all(sa), compare);
            vector<ll> nrank(n+1, 0);
            rep(i, 1, n+1) {
                nrank[sa[i]] = nrank[sa[i-1]] + compare(sa[i-1], sa[i]);
            }
            rank = nrank;
        }
        return sa;
    }
    vector<ll> make_lcp() {
        assert(sa.size() > 0);
        lcp.assign(sa.size(), 0);
        vector<ll> rank(n+1);
        rep(i, n+1) rank[sa[i]] = i;
        ll h = 0;
        rep(i, n) {
            if (h > 0) --h;
            assert(rank[i] > 0);
            for (ll j = sa[rank[i]-1]; j + h < n && i + h < n; h++) {
                if (str[j+h] != str[i+h]) break;
            }
            lcp[rank[i]-1] = h;
        }
        return lcp;
    }
    vector<ll> search(const string& s) {
        assert(lcp.size() > 0);
        ll l = -1, r = -1;
        {
            ll lb = 0, ub = n+1;
            while (ub - lb > 1) {
                ll mid = (lb + ub) / 2;
                if (str.substr(sa[mid], s.size()) >= s) {
                    ub = mid;
                }
                else {
                    lb = mid;
                }
            }
            l = ub;
        }
        {
            ll lb = 0, ub = n+1;
            while (ub - lb > 1) {
                ll mid = (lb + ub) / 2;
                if (str.substr(sa[mid], s.size()) > s) {
                    ub = mid;
                }
                else {
                    lb = mid;
                }
            }
            r = ub;
        }
        vector<ll> res;
        if (str.substr(sa[l], s.size()) == s) {
            rep(i, l, r) {
                res.pb(sa[i]);
            }
```

```
        }
        return res;
    }
};
```

SCC

```
void scc_dfs(ll v, vector<bool>& used, vector<ll>& vs, const vector< vector<ll> >& G) {
    used[v] = true;
    each(to, G[v]) {
        if (!used[to]) scc_dfs(to, used, vs, G);
    }
    vs.pb(v);
}
void scc_rdfs(ll v, ll k, vector<bool>& used, vector<ll>& cmp, const vector< vector<ll> >& rG) {
    used[v] = true;
    cmp[v] = k;
    each(to, rG[v]) {
        if (!used[to]) scc_rdfs(to, k, used, cmp, rG);
    }
}
// cmp が返る
// 同じ cmp は強連結成分
// cmp[i] < cmp[j] なら j から i に行けない
vector<ll> scc(const vector< vector<ll> >& G) {
    const ll n = G.size();
    vector<bool> used(n, false);
    vector<ll> vs;
    rep(i, n) {
        if (!used[i]) scc_dfs(i, used, vs, G);
    }
    used.assign(n, false);
    vector< vector<ll> > rG(n);
    rep(i, n) {
        each(to, G[i]) {
            rG[to].pb(i);
        }
    }
    vector<ll> res(n);
    ll k = 0;
    rrep(i, vs.size()) {
        if (!used[vs[i]]) scc_rdfs(vs[i], k++, used, res, rG);
    }
    return res;
}
vector< vector<ll> > get_scc_graph(const vector<ll>& cmp, const vector< vector<ll> >& G) {
    vector< vector<ll> > res(*max_element(all(cmp))+1);
    rep(i, G.size()) {
        each(to, G[i]) {
            if (cmp[i] != cmp[to]) {
                res[cmp[i]].pb(cmp[to]);
            }
        }
    }
    rep(i, res.size()) {
        sort(all(res[i]));
        res[i].erase(unique(all(res[i])), res[i].end());
    }
    return res;
}
```

Segment Manager

```
class SegmentManager {
    ll len;
    set<P> s;
    ll length(const P& p) {
        return p.second - p.first;
    }
    set<P>::iterator erase(const set<P>::iterator it) {
        len -= length(*it);
        return s.erase(it);
    }
    void insert(const P& p) {
        len += length(p);
        s.insert(p);
    }
    set<P>::iterator lb(ll l) {
        return s.lower_bound({l, l});
    }
    set<P>::iterator ub(ll l) {
        return lb(l+1);
    }
public:
    SegmentManager() : len(0) {}
```

```
    void add(ll l, ll r) {
        if (r <= l) return;
        erase(l, r);
        // merge right
        {
            auto it = lb(r);
            if (it != s.end() && it->first == r) {
                r = it->second;
                erase(it);
            }
        }
        // merge left
        {
            auto it = lb(l);
            if (it != s.begin()) {
                --it;
                if (it->second == l) {
                    l = it->first;
                    erase(it);
                }
            }
        }
        // add
        insert({l, r});
    }
    void erase(ll l, ll r) {
        if (r <= l) return;
        // cut left
        {
            auto it = lb(l);
            if (it != s.begin()) {
                --it;
                if (it->second > l) {
                    insert({it->first, l});
                    if (it->second > r) {
                        insert({r, it->second});
                        erase(it);
                        return;
                    }
                    erase(it);
                }
            }
        }
        auto it = lb(l);
        auto itr = lb(r);
        while (it != itr) {
            if (it->second > r) {
                insert({r, it->second});
                it = erase(it);
                break;
            }
            it = erase(it);
        }
    }
    bool is_in(ll l, ll r) {
        assert(r >= l);
        auto it = ub(l);
        if (it == s.begin()) return false;
        --it;
        return it->first <= l && r <= it->second;
    }
    bool is_in(ll pos) { return is_in(pos, pos+1); }
    ll length() { return len; }
    ll count() { return s.size(); }
    void out() {
        each(p, s) {
            cout << "[" << p.first << ", " << p.second << ")" << endl;
        }
    }
};
```

Sigma

```
// sigma[0,N) floor(n*num/den)
ll sigma(ll num, ll den, ll N) {
    if (num == 0) return 0;
    if (num >= den) {
        return num/den * N*(N-1)/2 + sigma(num%den, den, N);
    }
    else {
        ll nN = num*(N-1)/den;
        return N * (nN+1) - sigma(den, num, nN+1) - N;
    }
}
```

Segment Tree

```cpp
// SegmentTree<int> seg(n, 0x7FFFFFFF, [](int a, int b){return min(a, b);});
template <class T>
class SegmentTree {
    using func_t = function<T(T, T)>;
    const int sz, n;
    const T id;
    func_t merge;
    vector<T> data;
    int expand(int n) const { return n == 1 ? n : expand((n + 1) / 2) * 2; }
public:
    SegmentTree(const vector<T> &init, T id, func_t merge) :
        sz(init.size()), n(expand(sz)), id(id), data(n * 2, id), merge(merge) {
        copy(begin(init), end(init), begin(data)+n);
        RREP(i, n) {
            data[i] = merge(data[i * 2 + 0], data[i * 2 + 1]);
        }
    }
    int size() const { return sz; }
    void update(int p, T val) {
        assert (0 <= p && p < sz);
        data[p += n] = val;
        while (p /= 2) data[p] = merge(data[p * 2], data[p * 2 + 1]);
    }
    T find(int l, int r) const {
        assert (0 <= l && l <= r && r <= sz);
        l += n; r += n;
        T res1 = id, res2 = id;
        while (l != r) {
            if (l % 2) res1 = merge(res1, data[l++]);
            if (r % 2) res2 = merge(data[--r], res2);
            l /= 2; r /= 2;
        }
        return merge(res1, res2);
    }
};
```

Starry Sky Tree

```cpp
class StarrySkyTree {
private:
    ll base;
    vector<ll> s;
    vector<ll> mn;
    void update_mn(ll n) {
        if (n == 0) return;
        mn[n] = min(get_min(n*2), get_min(n*2+1));
    }
    void add(ll l, ll r, ll n, ll L, ll R, ll val) {
        if (r <= L || R <= l) return;
        if (L <= l && r <= R) {
            s[n] += val;
            return;
        }
        ll m = (l + r) / 2;
        add(l, m, n*2, L, R, val);
        add(m, r, n*2+1, L, R, val);
        mn[n] = min(get_min(n*2), get_min(n*2+1));
    }
    ll get_min(ll n) {
        return min(linf, mn[n] + s[n]);
    }
    ll get_min(ll l, ll r, ll n, ll L, ll R) {
        if (r <= L || R <= l) return linf;
        if (L <= l && r <= R) return get_min(n);
        ll m = (l + r) / 2;
        ll res = min(get_min(l, m, n*2, L, R), get_min(m, r, n*2+1, L, R));
        return min(linf, res+s[n]);
    }
    void get_min_pos(ll l, ll r, ll n, ll L, ll R, ll sum, vector<ll>& res) {
        if (r <= L || R <= l) return;
        if (mn[n] + s[n] + sum > 0) return;
        assert(l < r);
        if (r - l == 1) {
            res.pb(n-base);
        }
        else {
            ll m = (l + r) / 2;
            get_min_pos(l, m, n*2, L, R, sum+s[n], res);
            get_min_pos(m, r, n*2+1, L, R, sum+s[n], res);
        }
    }
public:
    StarrySkyTree(ll n) {
```

```
        for (base = 1; base < n; base <<= 1);
        s = vector<ll>(base*2, 0);
        mn = vector<ll>(base*2, 0);
    }
    void add(ll l, ll r, ll val) {
        add(0, base, 1, l, r, val);
    }
    ll get_min(ll l, ll r) {
        assert(0 <= l && l < r && r <= base);
        return get_min(0, base, 1, l, r);
    }
    vector<ll> get_min_pos(ll l, ll r) {
        ll min_value = get_min(l, r);
        vector<ll> res;
        get_min_pos(0, base, 1, l, r, -min_value, res);
        return res;
    }
};
```

Union Find

```
class UnionFind {
    vector<ll> par, h, sz;
public:
    UnionFind(ll size) : par(size, 0), h(size, 0), sz(size, 1) {
        rep(i, size) par[i] = i;
    }
    void unite(ll u, ll v) {
        u = root(u), v = root(v);
        if (u == v) return;
        if (h[u] < h[v]) {
            par[u] = v;
            sz[v] += sz[u];
        }
        else {
            par[v] = u;
            sz[u] += sz[v];
        }
        if (h[u] == h[v]) ++h[u];
    }
    ll size(ll v) {
        return sz[root(v)];
    }
    bool isUnited(ll u, ll v) {
        return root(u) == root(v);
    }
    ll root(ll v) {
        if (par[v] == v) return v;
        return par[v] = root(par[v]);
    }
};
```