极角排序

```
bool compare(PII a, PII b)
{
    auto get = [](PII a)
    {
        if(a.x < 0 && a.y < 0)    return 1;
        if(a.x >= 0 && a.y < 0)    return 2;
        if(a.x == 0 && a.y == 0)    return 3;
        if(a.x > 0 && a.y >= 0)    return 4;
        return 5;
    };

    int qa = get(a), qb = get(b);
    if(qa != qb)    return qa < qb;

    // if(a * b == 0)
    //     return get_len(a) < get_len(b);

    return a * b > 0;
}
```

凸包

```
void andrew(int* p, int* stk, int& top, int n)
{
    sort(p, p + n);
    for(int i = 0; i < n; i ++)
    {
        while(top >= 1 && sign(area(p[stk[top - 1]], p[stk[top]], p[i])) <= 0)
        {
            if(sign(area(p[stk[top - 1]], p[stk[top]], p[i])) < 0)
                used[stk[top --]] = false;
            else
                top --;
        }
        stk[++ top] = i, used[i] = true;
    }

    used[0] = false;
    for(int i = n - 1; i >= 0; i --)
    {
        if(used[i])   continue;

        while(top >= 1 && sign(area(p[stk[top - 1]], p[stk[top]], p[i])) <= 0)
            top --;
        stk[++ top] = i;
    }
}
```

```
void andrew(PII* p, int& n, PII* s)     //p：原数组，s：凸包数组
{
    sort(p + 1, p + n + 1);

    int cnt = 0;
    s[++ cnt] = p[1];

    for(int i = 2; i <= n; i ++)
    {
        while(cnt > 1 && area(s[cnt - 1], s[cnt], p[i]) <= 0)   cnt--;
        s[++ cnt] = p[i];
    }

    int t = cnt;
    for(int i = n - 1; i >= 1; i --)
    {
        while(cnt > t && area(s[cnt - 1], s[cnt], p[i]) <= 0)   cnt--;
        s[++ cnt] = p[i];
    }
    n = cnt - 1;    //第一个点加了两次，别忘了-1
}
```

闵可夫斯基和

```
//s，h:凸包数组，G:闵可夫斯基和数组
void Minkowski(PII* s, int n, PII* h, int m, PII* G, int tot)
{
    for(int i = 1; i <= n; i ++)
        a[i] = s[i % n + 1] - s[i];
    for(int i = 1;i <= m;i++)
        b[i] = h[i % m + 1] - h[i];

    int i = 1, j = 1;
    tot = 1, G[tot] = s[1] + h[1];

    while(i <= n && j <= m)
        tot ++, G[tot] = G[tot - 1] + (a[i] * b[j] >= 0 ? a[i ++] : b[j ++]);

    while(i <= n) tot++, G[tot] = G[tot - 1] + a[i ++];
    while(j <= m) tot++, G[tot] = G[tot - 1] + b[j ++];
}
```

半平面交

```
bool compare(const Line& a, const Line& b)
{
    double A = a.angle, B = b.angle;
    if(!cmp(A, B))   return sign(area(a.x, a.y, b.y)) < 0;
    return A < B;
}

PDD get_line_intersection(const Line& a, const Line& b)
{
    return get_line_intersection(a.st, a.ed - a.st, b.st, b.ed - b.st);
}

bool on_right(const Line& a, const Line& b, const Line& c)
{
    PDD o = get_line_intersection(b, c);
    return sign(area(a.st, a.ed, o)) <= 0;
}

double half_plane_intersection(Line& line, int n, int* q, int& hh, int& tt)
{
    sort(line, line + n, compare);

    for(int i = 0; i < n; i ++)
    {
        if(i && !cmp(line[i].angle, line[i - 1].angle))   continue;
        while(hh < tt && on_right(line[i], line[q[tt - 1]], line[q[tt]]))   tt --;
        while(hh < tt && on_right(line[i], line[q[hh]], line[q[hh + 1]]))   hh ++;
        q[++ tt] = i;
    }
    while(hh < tt && on_right(line[q[hh]], line[q[tt - 1]], line[q[tt]]))   tt --;
    while(hh < tt && on_right(line[q[tt]], line[q[hh]], line[q[hh + 1]]))   hh ++;
    q[++ tt] = q[hh];
}
```

最小圆覆盖

```
pair<PDD, PDD> get_line(PDD a, PDD b)          //ab的中垂线
{
    return { (a + b) / 2, rotate(b - a, PI / 2) };
}

Circle get_circle(PDD a, PDD b, PDD c)          //三点确定圆
{
    auto u = get_line(a, b), v = get_line(a, c);
    PDD o = get_line_intersection(u.x, u.y, v.x, v.y);
    return { o, get_dist(o, a) };
}

void get_min_Circle(PDD* p, int n)
{
    random_shuffle(p, p + n);

    Circle c = { p[0], 0 };

    for(int i = 1; i < n; i ++)
        if(cmp(c.r, get_dist(c.o, p[i])) < 0)
        {
            c = { p[i], 0 };
            for(int j = 0; j < i; j ++)
                if(cmp(c.r, get_dist(c.o, p[j])) < 0)
                {
                    c = { (p[i] + p[j]) / 2, get_dist(p[i], p[j]) / 2 };
                    for(int k = 0; k < j; k ++)
                        if(cmp(c.r, get_dist(c.o, p[k])) < 0)
                            c = get_circle(p[i], p[j], p[k]);
                }
        }
}
```

三维凸包

```
double rand_eps()
{
    return ((double)rand() / RAND_MAX - 0.5) * eps;
}

struct Point
{
    double x, y, z;
    void shake()
    {
        x += rand_eps(), y += rand_eps(), z += rand_eps();
    }
}p[maxn];

struct Plane
{
    int v[3];

    Point norm()
    {
        return (p[v[1]] - p[v[0]]) * (p[v[2]] - p[v[0]]);
    }
    double area()
    {
        return norm().len() / 2;
    }
    bool above(Point a)
    {
        return (norm() & (a - p[v[0]])) >= 0;
    }
}plane[2 * maxn], np[2 * maxn];

void get_convex_3d(Plane* plane, int n, Plane* np, int m, bool g[][N])
{
    plane[m ++] = { 0, 1, 2 };
    plane[m ++] = { 2, 1, 0 };

    for(int i = 3; i < n; i ++)
    {
        int cnt = 0;
        for(int j = 0; j < m; j ++)
        {
            bool t = plane[j].above(p[i]);
            if(!t)   np[cnt ++] = plane[j];
            for(int k = 0; k < 3; k ++)
                g[plane[j].v[k]][plane[j].v[(k + 1) % 3]] = t;
        }

        for(int j = 0; j < m; j ++)
            for(int k = 0; k < 3; k ++)
            {
                int a = plane[j].v[k], b = plane[j].v[(k + 1) % 3];
                if(g[a][b] && !g[b][a])
                    np[cnt ++] = { a, b, i };
            }

        m = cnt;
        for(int j = 0; j < m; j ++)
            plane[j] = np[j];
    }
}
```

旋转卡壳

最小矩形覆盖

```
double project(PDD a, PDD b, PDD c)      //ac向量在ab向量投影大小
{
    return (b - a) & (c - a) / get_len(b - a);
}
```

```
void rotating_calipers()
{
    for(int i = 0, a = 1, b = 2, c = 2; i < top; i ++)
    {
        PDD d = p[stk[i]], e = p[stk[i + 1]];
        while(cmp(area(d, e, p[stk[b]]), area(d, e, p[stk[b + 1]])) < 0)
            b = (b + 1) % top;
        while(cmp(project(d, e, p[stk[a]]), project(d, e, p[stk[a + 1]])) < 0)
            a = (a + 1) % top;
        if(!i)   c = b;
        while(cmp(project(d, e, p[stk[c]]), project(d, e, p[stk[c + 1]])) > 0)
            c = (c + 1) % top;

        PDD x = p[stk[a]], y = p[stk[b]], z = p[stk[c]];
        double h = area(d, e, y) / get_len(e - d), w = (x - z) & (e - d) /
get_len(e - d);

        if(h * w < min_area)
        {
            min_area = h * w;
            ans[0] = d + norm(e - d) * project(d, e, x);
            ans[3] = d + norm(e - d) * project(d, e, z);
            PDD u = rotate(norm(e - d), -PI / 2);
            ans[1] = ans[0] + u * h;
            ans[2] = ans[3] + u * h;
        }
    }
}
```

三角剖分

圆与简单多边形面积并

```
double get_circle_line_intersection(PDD a, PDD b, PDD& pa, PDD& pb)
{
    PDD e = get_line_intersection(a, b - a, r, rotate(b - a, PI / 2));

    double mind = get_len(e);

    if(!on_segment(e, a, b))   mind = min(get_len(a), get_len(b));
    if(cmp(R, mind) <= 0)   return mind;
    double len = sqrt(R * R - get_len(e) * get_len(e));
    pa = e + norm(a - b) * len, pb = e + norm(b - a) * len;
    return mind;
}

double get_circle_trangle_union(PDD a, PDD b)
{
    double da = get_len(a), db = get_len(b);          //到圆心的距离
    if(cmp(R, da) >= 0 && cmp(R, db) >= 0)   return a * b / 2;

    PDD pa, pb;          //直线与圆的交点
    double mind = get_circle_line_intersection(a, b, pa, pb);         //圆心到线段
a,b的距离

    if(cmp(R, mind) <= 0)   return get_sector(a, b);
    if(cmp(R, da) >= 0)   return a * pb / 2 + get_sector(pb, b);
    if(cmp(R, db) >= 0)   return get_sector(a, pa) + pa * b / 2;
    return get_sector(a, pa) + pa * pb / 2 + get_sector(pb, b);
}

double get_circle_polygon_union(PDD* p, int n)      //圆心在原点处
{
    double ans = 0;
    for(int i = 0; i < n; i ++)
        ans += get_circle_trangle_union(p[i], p[(i + 1) % n]);
}
```

扫描线

三角形面积并

```
double line_area(double a, int side)          //直线a长度与三角形交集长度
{
    vector<PDD> Y;
    for(int i = 0; i < n; i ++)
    {
        if(cmp(p[i][0].x, a) > 0 || cmp(p[i][2].x, a) < 0)   continue;

        if(!cmp(p[i][0].x, a) && !cmp(p[i][1].x, a))
        {
            if(side)   Y.push_back({ p[i][0].y, p[i][1].y });
        }
        else if(!cmp(p[i][1].x, a) && !cmp(p[i][2].x, a))
        {
            if(!side)   Y.push_back({ p[i][1].y, p[i][2].y });
        }
        else
        {
            double d[3];
            int u = 0;
            for(int j = 0; j < 3; j ++)
            {
                PDD o = get_line_intersection(p[i][j], p[i][(j + 1) % 3] - p[i]
[j], { a, -INF }, { 0, INF * 2 });
                if(cmp(o.x, INF))
                    d[u ++] = o.y;
            }
            if(u)
            {
                sort(d, d + u);
                Y.push_back({ d[0], d[u - 1] });
            }
        }
    }
    if(!Y.size())   return 0;
    sort(Y.begin(), Y.end());

    double res = 0, st = Y[0].fi, ed = Y[0].se;
    for(int i = 1, len = Y.size(); i < len; i ++)
        if(Y[i].fi <= ed)   ed = max(ed, Y[i].se);
        else
        {
            res += ed - st;
            st = Y[i].fi, ed = Y[i].se;
        }
    res += ed - st;

    return res;
}

double range_area(double a, double b)
{
    return (line_area(a, 1) + line_area(b, 0)) * (b - a) / 2;
}

double triangle_sum_area()
{
     for(int i = 0; i < n; i ++)
        for(int j = i + 1; j < n; j ++)
            for(int x = 0; x < 3; x ++)
                for(int y = 0; y < 3; y ++)
                {
                    PDD o = get_line_intersection(p[i][x], p[i][(x + 1) % 3] - p[i][x], p[j][y], p[j][(y + 1) % 3] - p[j][y]);
                    if(cmp(o.x, INF))
                        X.push_back(o.x);
                }

    sort(X.begin(), X.end());
    X.erase(unique(X.begin(), X.end()), X.end());

    for(int i = 0, len = X.size(); i + 1 < len; i ++)
        ans += range_area(X[i], X[i + 1]);
}
```

自适应辛普森积分

圆的面积并

```
double f(double x)               //横坐标为x的直线与圆的交集长度
{
    int cnt = 0;
    for(int i = 0; i < n; i ++)
    {
        double X = abs(c[i].O.fi - x);
        if(cmp(X, c[i].r) < 0)
        {
            double y = sqrt(c[i].r * c[i].r - X * X);
            p[cnt ++] = { c[i].O.se - y, c[i].O.se + y };
        }
    }

    if(!cnt)   return 0;

    sort(p, p + cnt);
    double res = 0, st = p[0].fi, ed = p[0].se;
    for(int i = 1; i < cnt; i ++)
        if(cmp(p[i].fi, ed) <= 0)  ed = max(ed, p[i].se);
        else
        {
            res += ed - st;
            st = p[i].fi, ed = p[i].se;
        }

    res += ed - st;
    return res;
}

double simpson(double l, double r)
{
    double mid = (l + r) / 2;
    return (f(l) + f(mid) * 4 + f(r)) / 6 * (r - l);
}

double asr(double l, double r, double s)
{
    double mid = (l + r) / 2;
    double left = simpson(l, mid), right = simpson(mid, r);

    if(!cmp(left + right, s))   return left + right;
    return asr(l, mid, left) + asr(mid, r, right);
}
```

模拟退火

```
double rand(double l, double r)
{
    return ((double)rand() / RAND_MAX * (r - l)) + l;
}

double calc(PDD a)
{
    double res = 0;
    for(int i = 0; i < n; i ++)
        res += get_dist(a, p[i]);

    if(cmp(ans, res) > 0)
        ans = res, ans_x = a.x, ans_y = a.y;

    return res;
}

```

```
void simulate_anneal()
{
    double x = ans_x, y = ans_y;

    for(double t = 1e3; t > 1e-10; t *= 0.99995)
    {
        double nx = x + t * rand(-1, 1), ny = y +  t * rand(-1, 1);

        double a = calc({ x, y }), b = calc({ nx, ny }), dt = b - a;
        if(dt < 0)
            x = nx, y = ny, ans = b;
        else if(cmp(exp(-dt / t), rand(0, 1)) > 0)
            x = nx, y = ny;
    }
}

int main()
{
    srand((unsigned)time(NULL));

    while((double)clock() / CLOCKS_PER_SEC < 1.5)
        simulate_anneal();
}
```

三分

```
//找最小值
void Third_Sarch(double l, double r)
{
    while(r - l >= eps)
    {
        double A = l + (r - l) / 3, B = l + 2 * (r - l) / 3;
        if(calc(A) <= calc(B))  r = B;
        else    l = A;
    }
}

//找最大值
void Third_Sarch(double l, double r)
{
    while(r - l >= eps)
    {
        double A = l + (r - l) / 3, B = l + 2 * (r - l) / 3;
        if(calc(A) >= calc(B))  r = B;
        else    l = A;
    }
}
```