# 计算几何补充

## 皮克定理

皮克定理是指一个计算点阵中顶点在格点上的多边形面积公式，该公式可以表示为S=a+b÷2－1(没有括号)，其中a表示多边形内部的点数，b表示多边形落在格点边界上的点数，S表示多边形的面积。

## 最小包含点矩形

```
const double eps = 1e-8;
const int N = 4004;
int sign(double d)
{
    return d < -eps ? -1 : (d > eps);
}
struct point
{
    double x, y;
    point operator-(point d){
        point dd;
        dd.x = this->x - d.x;
        dd.y = this->y - d.y;
        return dd;
    }
    point operator+(point d){
        point dd;
        dd.x = this->x + d.x;
        dd.y = this->y + d.y;
        return dd;
    }
    void read(){ scanf("%lf%lf", &x, &y); }
}ps[N];

int n, cn;

double dist(point d1, point d2)
{
    return sqrt(pow(d1.x - d2.x, 2.0) + pow(d1.y - d2.y, 2.0));
}

double dist2(point d1, point d2)
{
    return pow(d1.x - d2.x, 2.0) + pow(d1.y - d2.y, 2.0);
}

bool cmp(point d1, point d2)
{
```

```
    return d1.y < d2.y || (d1.y == d2.y && d1.x < d2.x);
}

//st1-->ed1叉乘st2-->ed2的值
double xmul(point st1, point ed1, point st2, point ed2)
{
    return (ed1.x - st1.x) * (ed2.y - st2.y) - (ed1.y - st1.y) * (ed2.x - st2.x);
}

double dmul(point st1, point ed1, point st2, point ed2)
{
    return (ed1.x - st1.x) * (ed2.x - st2.x) + (ed1.y - st1.y) * (ed2.y - st2.y);
}

//多边形类
struct poly
{
    static const int N = 50005; //点数的最大值
    point ps[N+5]; //逆时针存储多边形的点,[0,pn-1]存储点
    int pn;  //点数
    poly() { pn = 0; }
    //加进一个点
    void push(point tp)
    {
        ps[pn++] = tp;
    }
    //第k个位置
    int trim(int k)
    {
        return (k+pn)%pn;
    }
    void clear(){ pn = 0; }
};

//返回含有n个点的点集ps的凸包
poly graham(point* ps, int n)
{
    std::sort(ps, ps + n, cmp);
    poly ans;
    if(n <= 2){
        for(int i = 0; i < n; i++)
        {
            ans.push(ps[i]);
        }
        return ans;
    }
    ans.push(ps[0]);
    ans.push(ps[1]);
    point* tps = ans.ps;
```

```
    int top = -1;
    tps[++top] = ps[0];
    tps[++top] = ps[1];
    for(int i = 2; i < n; i++)
    {
        while(top > 0 && xmul(tps[top - 1], tps[top], tps[top - 1], ps[i]) <= 0) top--;
        tps[++top] = ps[i];
    }

    int tmp = top;  //注意要赋值给tmp!

    for(int i = n - 2; i >= 0; i--)
    {
        while(top > tmp && xmul(tps[top - 1], tps[top], tps[top - 1], ps[i]) <= 0) top-
-;
        tps[++top] = ps[i];
    }
    ans.pn = top;
    return ans;
}
//求点p到st->ed的垂足，列参数方程
point getRoot(point p, point st, point ed)
{
    point ans;
    double u=((ed.x-st.x)*(ed.x-st.x)+(ed.y-st.y)*(ed.y-st.y));
    u = ((ed.x-st.x)*(ed.x-p.x)+(ed.y-st.y)*(ed.y-p.y))/u;
    ans.x = u*st.x+(1-u)*ed.x;
    ans.y = u*st.y+(1-u)*ed.y;
    return ans;
}

//next为直线(st,ed)上的点，返回next沿(st,ed)右手垂直方向延伸l之后的点
point change(point st, point ed, point next, double l)
{
    point dd;
    dd.x = -(ed - st).y;
    dd.y = (ed - st).x;
    double len = sqrt(dd.x * dd.x + dd.y * dd.y);
    dd.x /= len, dd.y /= len;
    dd.x *= l, dd.y *= l;
    dd = dd + next;
    return dd;
}

//求含n个点的点集ps的最小面积矩形，并把结果放在ds(ds为一个长度是4的数组即可,ds中的点是逆时针的)中，并
返回这个最小面积。
double getMinAreaRect(point* ps, int n, point* ds)
{
    int cn, i;
```

```
    double ans;
    point* con;
    poly tpoly = graham(ps, n);
    con = tpoly.ps;
    cn = tpoly.pn;
    if(cn <= 2)
    {
        ds[0] = con[0]; ds[1] = con[1];
        ds[2] = con[1]; ds[3] = con[0];
        ans=0;
    }
    else
    {
        int  l, r, u;
        double tmp, len;
        con[cn] = con[0];
        ans = 1e40;
        l = i = 0;
        while(dmul(con[i], con[i+1], con[i], con[l])
            >= dmul(con[i], con[i+1], con[i], con[(l-1+cn)%cn]))
        {
            l = (l-1+cn)%cn;
        }
        for(r=u=i = 0; i < cn; i++){
            while(xmul(con[i], con[i+1], con[i], con[u])
                <= xmul(con[i], con[i+1], con[i], con[(u+1)%cn]))
            {
                u = (u+1)%cn;
            }
            while(dmul(con[i], con[i+1], con[i], con[r])
                <= dmul(con[i], con[i+1], con[i], con[(r+1)%cn]))
            {
                r = (r+1)%cn;
            }
            while(dmul(con[i], con[i+1], con[i], con[l])
                >= dmul(con[i], con[i+1], con[i], con[(l+1)%cn]))
            {
                l = (l+1)%cn;
            }
            tmp = dmul(con[i], con[i+1], con[i], con[r]) - dmul(con[i], con[i+1],
con[i], con[l]);
            tmp *= xmul(con[i], con[i+1], con[i], con[u]);
            tmp /= dist2(con[i], con[i+1]);
            len = xmul(con[i], con[i+1], con[i], con[u])/dist(con[i], con[i+1]);
            if(sign(tmp - ans) < 0)
            {
                ans = tmp;
                ds[0] = getRoot(con[l], con[i], con[i+1]);
                ds[1] = getRoot(con[r], con[i+1], con[i]);
```

```
                ds[2] = change(con[i], con[i+1], ds[1], len);
                ds[3] = change(con[i], con[i+1], ds[0], len);
            }
        }
    }
    return ans+eps;
}
```