旋转卡壳

Rotating Calipers

Idea：首先求出凸包，随后逆时针枚举边，利用三角形面积寻找最远点，容易发现最远点的轨迹也是逆时针的，该最远点可能是边的两个端点的对踵点对，由此可求出所有对踵点对。

Application：求凸包直径、宽度，凸包间最大、小距离，最小面积、周长外接矩形，洋葱、螺旋三角剖分，四边形剖分，合并凸包、凸包公切线、凸包交集、凸包临界切线、凸多边形矢量和，最薄横截带

Reference：链接

Complexity：$O(n)$（仅就旋转卡壳而言；事实上，由于一般需要先求凸包，复杂度是凸包的复杂度）

求凸包直径的平方

```
int ans;
void RotatingCalipers(int m, Point p[]){ // p[] = sta[], m = staid in ConvexHull()
    if(m == 2){
        ans = (int)( (p[1] - p[2]) * (p[1] - p[2]) );
        return;
    }
    p[m+1] = p[1];
    int ver = 2;
    for(int i = 1; i <= m; i++){
        while(TriangleArea(p[i], p[i+1], p[ver]) < TriangleArea(p[i], p[i+1], p[ver+1])){
            ver++;
            if(ver == m+1)    ver = 1;
            ans = max(ans, (int)max((p[ver] - p[i]) * (p[ver] - p[i]), (p[ver] - p[i+1]) * (p[ver] - p[i+1])));
        }
    }
}
```

最小矩形覆盖

```
struct MinRectangleCover{

    double minArea, minPeri;
    Point minAreaPoints[10], minPeriPoints[10];

    void cal(int i, int nxti, int ver, int j, int k, Point p[]){
        Point t[4];
        Vector v = p[nxti] - p[i], u = Normal(v);
        t[0] = GetLineIntersection(Line(p[i], v), Line(p[j], u));
        t[1] = GetLineIntersection(Line(p[j], u), Line(p[ver], v));
        t[2] = GetLineIntersection(Line(p[ver], v), Line(p[k], u));
        t[3] = GetLineIntersection(Line(p[k], u), Line(p[i], v));
        double area = fabs((t[1] - t[0]) ^ (t[0] - t[3]));
        if(cmp(area, minArea) < 0){
            minArea = area;
            minAreaPoints[0] = t[0], minAreaPoints[1] = t[1];
            minAreaPoints[2] = t[2], minAreaPoints[3] = t[3];
        }
        double peri = Length(t[1]-t[0]) + Length(t[0]-t[3]); peri *= 2;
        if(cmp(peri, minPeri) < 0){
            minPeri = peri;
            minPeriPoints[0] = t[0], minPeriPoints[1] = t[1];
            minPeriPoints[2] = t[2], minPeriPoints[3] = t[3];
        }
    }
    inline void Norm(int &x, int m){ ((x %= m) += m) %= m; if(x == 0)  x = m; }
    inline double func(int mid, int i, int nxti, Point p[], int m, int kind){
        Norm(mid, m);
        if(kind == 1)
```

```
            return (p[nxti]-p[i]) * (p[mid]-p[i]) / Length(p[nxti]-p[i]);
        else
            return (p[i]-p[nxti]) * (p[mid]-p[nxti]) / Length(p[i]-p[nxti]);
    }
    int tripartition(int l, int r, int i, int nxti, Point p[], int m, int kind){
        while(r < l)   r += m;
        int mid1 = l, mid2 = r;
        while(mid1 < mid2){
            mid1 = l + (r - l) / 3;
            mid2 = r - (r - l) / 3;
            // func(x) is a unimodal function
            if(func(mid1, i, nxti, p, m, kind) < func(mid2, i, nxti, p, m, kind))
                l = mid1 + 1;
            else    r = mid2 - 1;
        }
        return l;
    }
    // minimum rectangle covering the points p[]
    void solve(int m, Point p[]){
        minArea = minPeri = INF;
        int ver = 2;
        for(int i = 1; i <= m; i++){
            int nxti = i + 1; Norm(nxti, m);
            while(TriangleArea(p[i], p[nxti], p[ver]) < TriangleArea(p[i], p[nxti], p[ver+1]))
                ver++, Norm(ver, m);
            int l = nxti, r = ver;
            int j = tripartition(l, r, i, nxti, p, m, 1);
            l = ver, r = i;
            int k = tripartition(l, r, i, nxti, p, m, 2);
            Norm(k, m), Norm(j, m);
            cal(i, nxti, ver, j, k, p);
        }
    }

};
```