算法设计与分析 C5-F

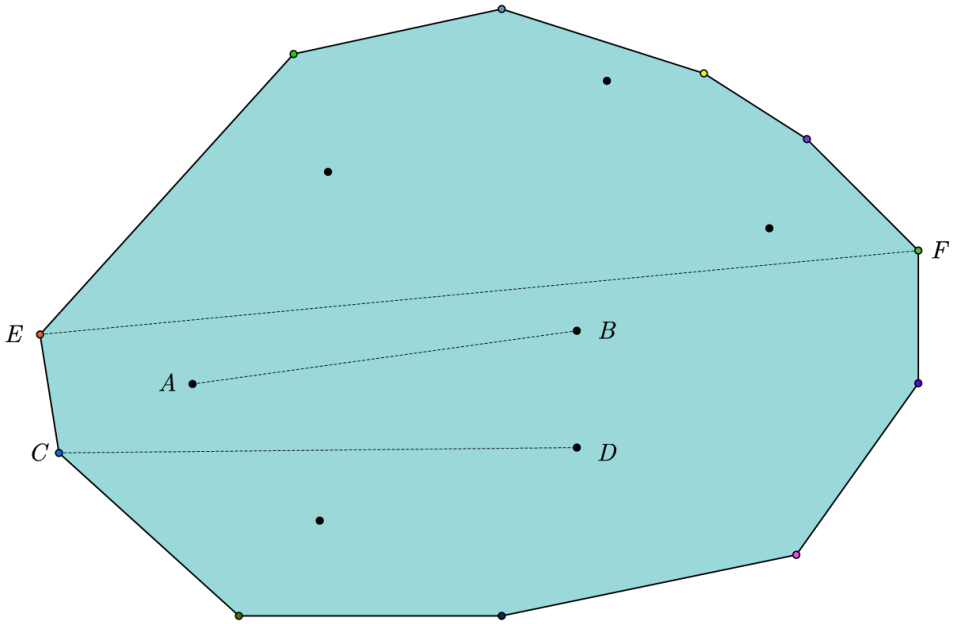
21377206 阮阳栋

题目描述

给定$\mathbb{R}^2$平面上的点集$S = \{P_1, P_2, ..., P_n\}$，求解点之间最大距离的平方，也就是$\max |P_iP_j|^2$，其中$|P_iP_j|$代表线段$P_iP_j$的长度。

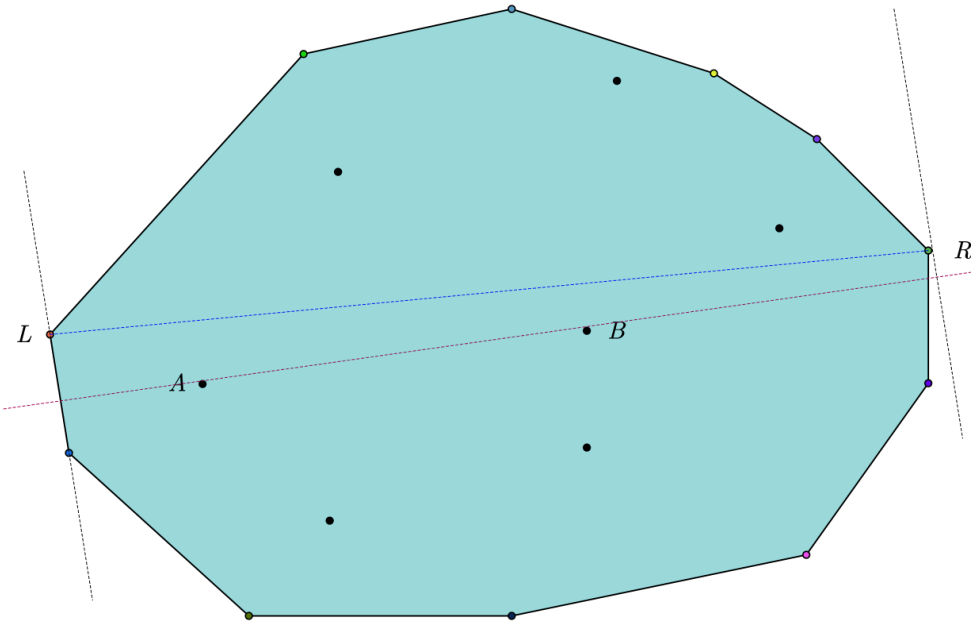
题目分析

分析点集可以想到先去求解它们的凸包，需要分析凸包中的三种点距离：内部点间距（如AB）、凸包端点和内部点距离（如CD）、凸包端点间距（如EF）。



凸包中的点距离

对于内部点来说，可以在其线段延长线上做一系列垂线，直到交到且仅交到一个凸包端点：



内部点间距不会是最大值

可以发现LR的长度一定比AB是要大的（可以把它们放在一个直角三角形中）。对于CD来说，也是同一道理，只要延长一段就可以发现有更大的距离。

所以最值一定是在凸包端点间取到。

题目求解

点的分布是未知的，所以采用Graham算法来求解凸包，此算法在C5的大多题目中都有所使用。

可以稍微进行优化，把相对于左下端点的一些共线的点删掉，减少后续的计算时间（`na` 存放的才是起始端点）：

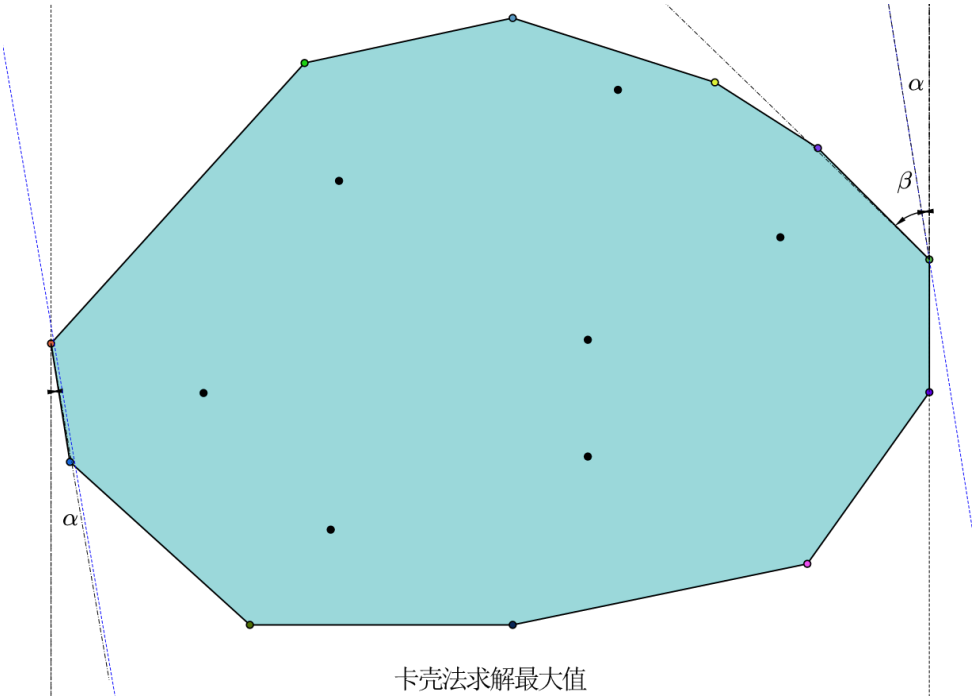
```
for (int i=1;i<=n;i++){
    if (i > 2 && !direc(a[1], a[i-1], a[i]))
        continue;
    na[cnt++] = a[i];
}
cnt--;
```

计算最大距离可以直接进行暴力（这已经可以通过测评了）：

```
ll result = 0;
for (int i=0;i-1<arrc1;i++){
    for (int j=i+1;j<arrc1;j++)
        result = max(result, len(ar[i], ar[j]));
}
```

可是题目中的点数可以到达 1e5 ，如果凸包上的点过于多怎么解决?

在点数较多时，可以模仿刚才的分析方法，对凸包进行卡壳：



卡壳法求解最大值

可以先选取左下点和右上点，作平行于y轴的两条直线，逆时针进行旋转。旋转时看下一条边相对于现在直线的角度，选取两条线的两个角的最小值，旋转后重新计算两条直线的距离。旋转一圈后便能得出最大距离。

时间复杂度

Graham算法的时间复杂度为$O(n\log n)$，如果使用暴力找最大值，复杂度为$O(h^2)$，如果使用卡壳法则为$O(h)$。

最终复杂度为$O(n\log n+h^2)$或者$O(n\log n+h)$。

核心代码

Graham算法：

```
bool graham(){
    pt p0 = a[1]; int id = 1;
    for (int i=2;i<=n;i++){
        if (a[i].y < p0.y || (a[i].y == p0.y && a[i].x < p0.x))
        p0 = a[i], id = i;
    }
    a[id] = a[1], a[1] = p0;
    cpp_stablesort(2, n);

    for (int i=1;i<=n;i++){
        if (i > 2 && !direc(a[1], a[i-1], a[i])) continue;
        na[cnt++] = a[i];
    }
    cnt--;

    if (cnt-1 < 2) return false;
    while (!s0.empty()) s0.pop();
    s0.push(na[1]), s0.push(na[2]), s0.push(na[3]);
    for (int i=4;i<=cnt;i++){
        while (1){
            pt iniTop = s0.top(), nextTop;
            s0.pop(), nextTop = s0.top(), s0.push(iniTop);
            if (direc(s0.top(), na[i], nextTop) >= 0) s0.pop();
            else break;
        }
        s0.push(na[i]);
    }
    return true;
}
```