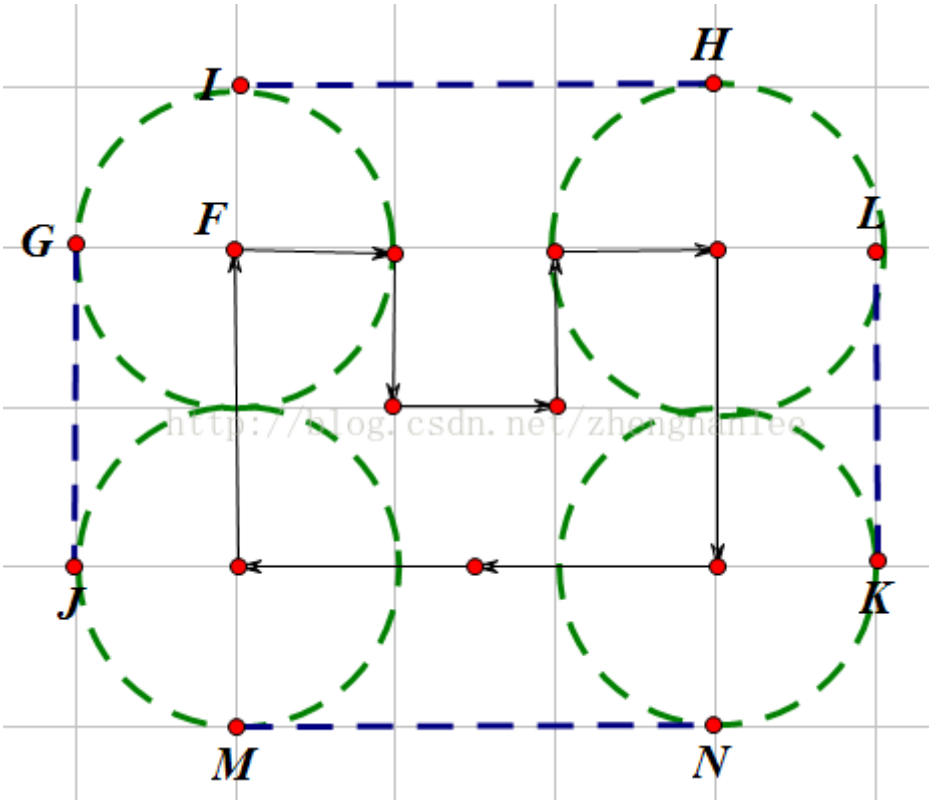
Solution - 计算几何

A Wall



Solution

题目大意:给定平面上多边形（坐标形式按顺序给出）定义为城堡，为城堡修建围墙，围墙距离多边形一定的距离L。求最小围墙周长。求凸包，再加上半径为L的圆即可。求凸包后结果为一个凸多边形，每条边向外拉伸L的距离即为围墙的直线部分，连接每条边需要半径为L的圆弧围墙，所有圆弧围墙拼在一起恰好是整圆。见上图。

Code

```
#include<iostream>
#include<algorithm>
#include<cmath>
using namespace std;
const int maxn=1e3+10;
#define pi acos(-1.0)
struct point{
    int x,y;
}ch[maxn],p[maxn];
bool cmp(point a,point b){
    if(a.x==b.x) return a.y<b.y;
    else return a.x<b.x;
}
int det(point a,point b,point c){//叉乘
    return (b.x-a.x)*(c.y-a.y)-(b.y-a.y)*(c.x-a.x);
}
```

```
int Andrew(point *p,int n,point *ch){//Andrew算法求凸包
    sort(p,p+n,cmp);
    int m=0;
    for(int i=0;i<n;i++){
        while(m>1&&det(ch[m-2],ch[m-1],p[i])<0)
            m--;
        ch[m++]=p[i];//ch保存凸包的坐标点
    }
    int k=m;
    for(int i=n-2;i>=0;i--){
        while(m>k&&det(ch[m-2],ch[m-1],p[i])<0)
            m--;
        ch[m++]=p[i];
    }
    if(n>1) m--;
    return m;
}
double dis(point a,point b){//两点间距离
    return sqrt(1.0*(b.x-a.x)*(b.x-a.x)+(b.y-a.y)*(b.y-a.y));
}
int main(){
    int n,L;
    double sum=0;
    cin>>n>>L;
    for(int i=0;i<n;i++)
        cin>>p[i].x>>p[i].y;
    int m=Andrew(p,n,ch);
    for(int i=0;i<m-1;i++)
        sum+=dis(ch[i],ch[i+1]);
    sum+=dis(ch[m-1],ch[0])+2*pi*L;
    printf("%.lf\n",sum);
    return 0;
}
```

B Beauty Contest

Solution

题目大意：给定平面上若干农场（坐标形式）求相距最远的一对农场的距离。最远点对的模板题，先求凸包，再在凸包上用旋转卡壳即可。
注意：这里的代码求凸包使用的是Graham-Scan的极角序法，注意与Andrew的求上下凸包的区别。题目要求输出的是距离的平方，因此可以全程使用整形。

Code

```
#include<bits/stdc++.h>
using namespace std;
typedef double db;
const int N=100001;
struct P{int x,y;} a[N],s[N];
int n,top=0,ans=0;
P operator-(P a,P b){
    return (P){a.x-b.x,a.y-b.y};
}
int operator*(P a,P b){
    return a.x*b.y-a.y*b.x;
```

```
}
int dis(P a){
    return a.x*a.x+a.y*a.y;
}
bool cmp(P x,P y){
    if(abs((x-a[1])*(y-a[1]))>0) return (x-a[1])*(y-a[1])>0;
    return dis(x-a[1])<dis(y-a[1]);
}
int S(P a,P b,P c){
    return abs((a-b)*(a-c))/2;
}
int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;i++)
        scanf("%d%d",&a[i].x,&a[i].y);
    for(int i=2;i<=n;i++)
        if(a[i].y<a[1].y||(a[i].y==a[1].y&&a[i].x<a[1].x))
            swap(a[1],a[i]);
    sort(a+2,a+n+1,cmp);
    top=2;s[1]=a[1];s[2]=a[2];
    for(int i=3;i<=n;i++){
        while(top>1&&(a[i]-s[top-1])*(s[top]-s[top-1])>=0) top--;
        s[++top]=a[i];
    }
    for(int i=0;i<top;i++)
        s[i]=s[i+1];
    n=top;
    for(int i=0,j=2;i<n;i++){
        while(S(s[i],s[(i+1)%n],s[j])<S(s[i],s[(i+1)%n],s[(j+1)%n])) j=(j+1)%n;
        ans=max(dis(s[i]-s[j]),ans);
    }
    printf("%d",ans);
    return 0;
}
```

C 改革春风吹满地

Solution

题目大意：多组数据，给定平面上多边形（坐标形式按顺序给出），求面积。利用向量叉乘得到的是平行四边形有效面积的性质，可将多边形拆分成有向三角形面积的和。

Code

```
#include <bits/stdc++.h>
int main(){
    int n, i;
    while (scanf("%d", &n) != EOF){
        if (n == 0) return 0;
        int x[100], y[100];
        double ans = 0;
        for (i = 1; i <= n; i++)
            scanf("%d %d", &x[i],&y[i]);
        x[n + 1] = x[1]; y[n + 1] = y[1];
        for (i = 1; i <= n; i++)
            ans += (x[i] * y[i + 1] - y[i] * x[i + 1]) / 2.0;
```

```
        printf("%.1f\n", ans);//保留1位
    }
    return 0;
}
```

D Intersecting Lines

Solution

题目大意：多组数据，每组给定两条直线（直线以过两定点形式给出），即8个值代表的四个点坐标，判断是相交/平行/重合。原理就是叉乘。利用叉乘可求出直线的方向向量关系，从而区分相交或是不相交。相交则求交点，套用公式即可。不相交则需要判断是重合还是平行，只需要连接一条直线的一个点到另一条直线两个点的向量再作判断即可，若重合，则这两个向量依然平行，若不平行，则说明原先的两条直线是平行而不是重合。画图即可理解。

Code

这里附上Kuangbin大神的代码，大家可以仔细研究研究。

```
/*************************************************************
 * Author        : kuangbin
 * Email         : kuangbin2009@126.com
 * Last modified : 2013-07-14 08:54
 * Filename      : POJ1269IntersectingLines.cpp
 * Description   :
 * ***********************************************************/

#include <bits/stdc++.h>
using namespace std;
const double eps = 1e-8;
int sgn(double x)
{
    if(fabs(x) < eps)return 0;
    if(x < 0)return -1;
    else return 1;
}
struct Point
{
    double x,y;
    Point(){}
    Point(double _x,double _y)
    {
        x = _x;y = _y;
    }
    Point operator -(const Point &b)const
    {
        return Point(x - b.x,y - b.y);
    }
    double operator ^(const Point &b)const
    {
        return x*b.y - y*b.x;
    }
    double operator *(const Point &b)const
    {
        return x*b.x + y*b.y;
    }
```

```
};
struct Line
{
    Point s,e;
    Line(){}
    Line(Point _s,Point _e)
    {
        s = _s;e = _e;
    }
    pair<Point,int> operator &(const Line &b)const
    {
        Point res = s;
        if(sgn((s-e)^(b.s-b.e)) == 0)
        {
            if(sgn((b.s-s)^(b.e-s)) == 0)
                return make_pair(res,0);//两直线重合
            else return make_pair(res,1);//两直线平行
        }
        double t = ((s-b.s)^(b.s-b.e))/((s-e)^(b.s-b.e));
        res.x += (e.x - s.x)*t;
        res.y += (e.y - s.y)*t;
        return make_pair(res,2);//有交点
    }
};
int main(){
    //freopen("in.txt","r",stdin);
    //freopen("out.txt","w",stdout);
    int T;
    scanf("%d",&T);
    double x1,y1,x2,y2,x3,y3,x4,y4;
    printf("INTERSECTING LINES OUTPUT\n");
    while(T--){
        scanf("%lf%lf%lf%lf%lf%lf%lf%lf",&x1,&y1,&x2,&y2,&x3,&y3,&x4,&y4);
        Line line1 = Line(Point(x1,y1),Point(x2,y2));
        Line line2 = Line(Point(x3,y3),Point(x4,y4));
        pair<Point,int> ans = line1 & line2;
        if( ans.second == 2)printf("POINT %.2lf
%.2lf\n",ans.first.x,ans.first.y);
        else if(ans.second == 0)printf("LINE\n");
        else printf("NONE\n");
    }
    printf("END OF OUTPUT\n");
    return 0;
}
```