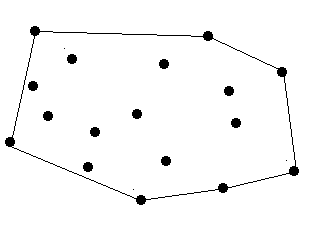
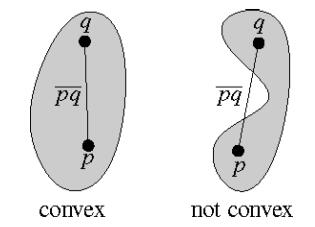
计算几何：凸包

极点：在一组点中，若沿着某一个点做直线，必然能找到一条直线使得所有点都在直线的另一侧，则这个点为极点。

凸包：给定二维平面上的点集，凸包就是将最外层的点连接起来构成的凸多边形，它能包含点集中所有的点



平面的一个子集S被称为是“凸”的，当且仅当对于任意两点p，s∈S，线段ps都完全属于S。（平面凸包定义）：



计算凸包：

Graham扫描法: 复杂度O(nlogn)

求得凸包中的点是按逆时针排序的

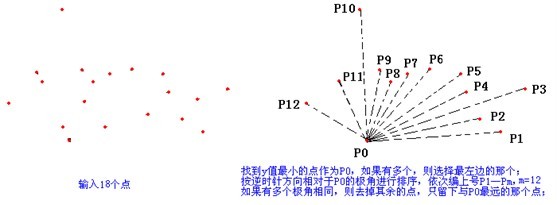
<=表示如果存在共线的极点，则只取端点

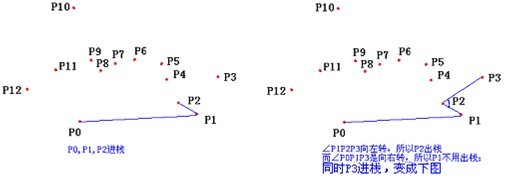
扫描方法有两种，可以用极角序（按极角排序）或坐标序（按x坐标排序，x坐标一样按y坐标排序，其中扫描上凸包是从大到小，扫描下凸包是从小到大）

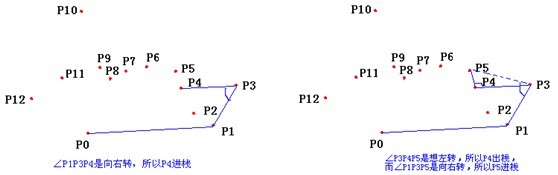
找到最左下的点(即y最小，y一样取最左边的)，赋给p[1]

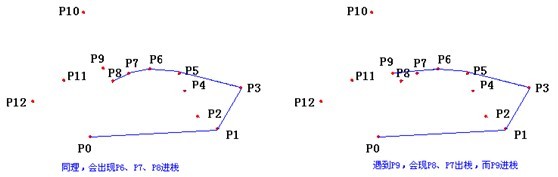
以p[1]作为极点，将所有点按极角排序，极角相同则按距离排序（去掉极角相同的，只留下距离最远的那个点）

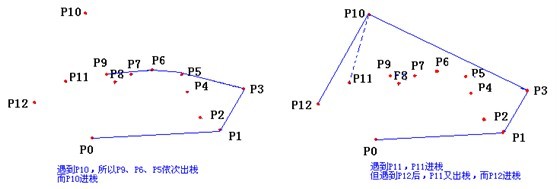
求凸包时while循环把发现不是凸包顶点的点移除出去。因为当逆时针遍历凸包时，我们应该在每个顶点向左转，因此当while循环发现在一个顶点处没有向左转时，就把该顶点移除出去。

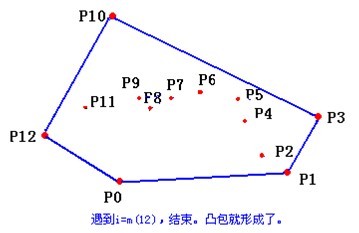












计算凸包周长：

hdu1392:Surround the Trees

//以左下点为极点求凸包周长

#include<bits/stdc++.h>

using namespace std;

struct point{

double x,y;

point friend operator -(point a,point b)

{return {a.x-b.x,a.y-b.y};}

}p[105],s[105];//p为所有点的集合,s为凸包点集

double dis(point a,point b)

{

point c=a-b;

return sqrt(c.x\*c.x+c.y\*c.y);

}

double cross(point a,point b)

{

return a.x\*b.y-a.y\*b.x;

}

//按照极角(polar angle)从小到大排序(以 p1为极点)

//极角相同的点按照到的距离从小到大排序。

int cmp(point a,point b)

{

double x=cross(a-p[1],b-p[1]);

if(x>0) return 1;

if(x==0&&dis(a,p[1])<dis(b,p[1])) return 1;

return 0;

}

double multi(point p1,point p2,point p3)//计算向量叉积

{

return cross(p2-p1,p3-p1);

}

int main()

{

int N;

while(scanf("%d",&N),N)

{

for(int i=1;i<=N;i++) cin>>p[i].x>>p[i].y;

if(N==1)

{

printf("0.00\n");

continue;

}

else if(N==2)

{

printf("%.2lf\n",dis(p[1],p[2]));

continue;

}

int k=1;

for(int i=2;i<=N;i++)

if(p[i].y<p[k].y||(p[i].y==p[k].y&&p[i].x<p[k].x))k=i;

swap(p[1],p[k]);//找到处于最左下方的点,赋给p[1]

sort(p+2,p+1+N,cmp);//对点按极角排序,p[1]为极点

s[1]=p[1];

s[2]=p[2];

int cnt=2;

//while循环把发现不是凸包顶点的点移除出去,因为当逆时针遍历凸包时，我们应该在每个顶点向左转

//因此当while循环发现在一个顶点处没有向左转时，就把该顶点移除出去。

for(int i=3;i<=N;i++)//求凸包

{

while(cnt>=2&&multi(s[cnt-1],s[cnt],p[i])<=0) cnt--;

s[++cnt]=p[i];

}

double sum=0;

for(int i=1;i<cnt;i++)//求凸包周长

sum+=dis(s[i],s[i+1]);

printf("%.2f\n",sum+dis(s[1],s[cnt]));

/\*

double area=0;

for(int i=2;i<=cnt-1;i++)//求凸包面积，这里因为就是凸多边形所以可以用某个顶点作为顶点分割三角形

area+=fabs(multi(s[1],s[i],s[i+1]));

printf("%d\n",int(area/100));

\*/

/\*标准求多边形面积,别漏了最后一个和起点的叉积

for(int i=1;i<=cnt;i++)//求凸包面积

area+=cross(s[i],s[i==cnt?1:i+1]);

if(area<0)

area=-area;

\*/

}

return 0;

}

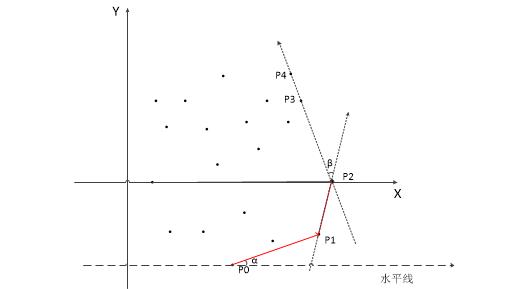
jarvis步进法（凸包）

思路：

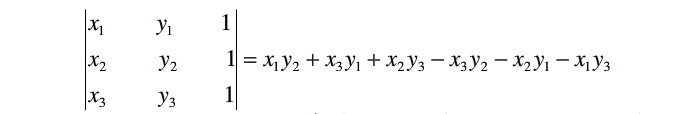
1.先找到纵坐标最小点p0入栈,遍历剩下的点，找到与水平方向夹角最小的点p1入栈

2.遍历所有点找到与栈顶两个点连线夹角最小的点pn入栈，重复该过程，直道找不出下一个pn

3.栈里的所有点就是凸包上的点



关于怎么找最小角这一步我感觉完全可以替换成每次先把未入栈的一个点入栈，然后遍历剩下所有点，如果某点在栈顶两点连线的右方，用该点顶替栈顶点，每次遍历完所有的点，标记一下栈顶的点，如果栈顶点被标记过，舍弃栈顶元素，剩下的点就是凸包上的所有点。



该式大于零第三个点在前两个点的左方，否则在右方。

以poj1113 Wall为例看一下代码

#include <iostream>

#include <cstring>

#include <cstdio>

#include <cmath>

#include <cstdlib>

#include <algorithm>

#define inf 0x3f3f3f3f

#define PI acos(-1.0)

#define eps 1e-9

using namespace std;

struct node

{

double x, y;

};

node p[1100];

int s[1100];

int vis[1100];

int top;

double multi(node p1, node p2, node p3)

{

return p1.x \* p2.y + p3.x \* p1.y + p2.x \* p3.y - p3.x \* p2.y - p2.x \* p1.y - p1.x \* p3.y;

}

double dist(node a, node b)

{

return sqrt(pow(a.x-b.x,2) + pow(a.y-b.y,2));

}

void jarvis(int n)

{

node minp;

int i, m;

top = 0;

minp.x = inf;

minp.y = inf;

for(i = 0;i < n;i++)

{

if(p[i].y < minp.y || (p[i].y == minp.y && p[i].x < minp.x))

{

minp = p[i];

m = i;

}

}

s[top++] = m;

vis[m] = 1;

m = top;

int flag = 1;

while(flag)

{

flag = 0;

for(i = 0;i < n;i++)

{

if(top <= m)

{

if(vis[i])

continue;

s[top++] = i;

flag = 1;

continue;

}

if(multi(p[s[top-2]],p[s[top-1]],p[i]) < 0 || (abs(multi(p[s[top-2]],p[s[top-1]],p[i]) == 0) && dist(p[s[top-2]],p[i]) < dist(p[s[top-2]],p[s[top-1]]) && !vis[i]))

{

s[top-1] = i;

flag = 1;

}

}

if(vis[s[top-1]])

break;

vis[s[top-1]] = 1;

m = top;

}

}

int main()

{

int n, i;

double d, sum = 0;

cin>>n>>d;

for(i = 0;i < n;i++)

cin>>p[i].x>>p[i].y;

jarvis(n);

for(i = 0;i < top;i++)

{

sum += dist(p[s[i]],p[s[(i+1)%top]]);

}

printf("%.0lf\n",sum + 2.0 \* PI \* d);

return 0;

}