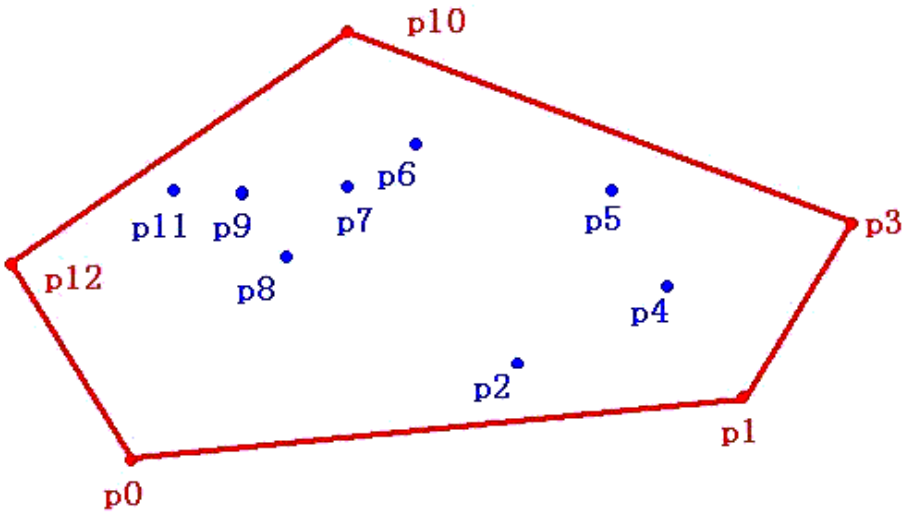
计算二维凸包的 Graham_Scan 算法

凸包的概念：

点集 Q 的凸包(convex hull)是指一个最小凸多边形，满足 Q 中的点或者在多边形边上或者在其内。下图中由红色线段表示的多边形就是点集 Q={p0,p1,...p12} 的凸包。



凸包的求法：

现在已经证明了凸包算法的时间复杂度下界是 O(n*logn)，但是当凸包的顶点数 h 也被考虑进去的话，Krikpatrick 和 Seidel 的剪枝搜索算法可以达到 O(n*logh)，在渐进意义下达到最优。最常用的凸包算法是 Graham 扫描法和 Jarvis 步进法。本文只简单介绍一下 Graham 扫描法，其正确性的证明和 Jarvis 步进法的过程大家可以参考《算法导论》。

对于一个有三个或以上点的点集 Q，Graham 扫描法的过程如下：

令 p0 为 Q 中 Y-X 坐标排序下最小的点；

设<p1,p2,...pm>为对其余点按以 p0 为中心的极角逆时针排序所得的点集（如果有多个点有相同的极角，除了距 p0 最远的点外全部移除；

压 p0 进栈 S；

压 p1 进栈 S；

压 p2 进栈 S；

for i ← 3 to m

{

do while{由 S 的栈顶元素的下一个元素、S 的栈顶元素以及 pi 构成的折线段不拐向左侧}对 S 弹栈;

压 pi 进栈 S;

}

return S;

此过程执行后，栈 S 由底至顶的元素就是 Q 的凸包顶点按逆时针排列的点序列。需要注意的是，我们对点按极角逆时针排序时，并不需要真正求出极角，只需要求出任意两点的次序就可以了。而这个步骤可以用前述的矢量叉积性质实现。

程序示例

```
#include <iostream>
#include <cmath>
using namespace std;

/*
PointSet[]：输入的点集
ch[]：输出的凸包上的点集，按照逆时针方向排列
n：PointSet中的点的数目
len：输出的凸包上的点的个数
*/

struct Point
{
    float x,y;
};

//小于,说明向量p0p1的极角大于p0p2的极角
float multiply(Point p1,Point p2,Point p0)
{
    return((p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y));
}

float dis(Point p1,Point p2)
{
    return(sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y)));
}
```

```
void Graham_scan(Point PointSet[],Point ch[],int n,int &len)
{
    int i,j,k=0,top=2;
    Point tmp;

    //找到最下且偏左的那个点
    for(i=1;i<n;i++)
    if
((PointSet[i].y<PointSet[k].y)||((PointSet[i].y==PointSet[k].y)&&(PointSet[i].x<Poi
ntSet[k].x)))
        k=i;
    //将这个点指定为PointSet[0]
    tmp=PointSet[0];
    PointSet[0]=PointSet[k];
    PointSet[k]=tmp;

    //按极角从小到大,距离偏短进行排序
    for (i=1;i<n-1;i++)
    {
        k=i;
        for (j=i+1;j<n;j++)
            if( (multiply(PointSet[j],PointSet[k],PointSet[0])>0)
                    ||((multiply(PointSet[j],PointSet[k],PointSet[0])==0)

    &&(dis(PointSet[0],PointSet[j])<dis(PointSet[0],PointSet[k])) )
                k=j;//k保存极角最小的那个点,或者相同距离原点最近
        tmp=PointSet[i];
        PointSet[i]=PointSet[k];
        PointSet[k]=tmp;
    }
    //第三个点先入栈
    ch[0]=PointSet[0];
    ch[1]=PointSet[1];
    ch[2]=PointSet[2];
    //判断与其余所有点的关系
    for (i=3;i<n;i++)
    {
        //不满足向左转的关系,栈顶元素出栈
        while(multiply(PointSet[i],ch[top],ch[top-1])>=0) top--;
        //当前点与栈内所有点满足向左关系,因此入栈.
        ch[++top]=PointSet[i];
    }
```

```
    len=top+1;
}

const int maxN=1000;
Point PointSet[maxN];
Point ch[maxN];
int n;
int len;

int main()
{
    int n=5;
    float x[]={0,3,4,2,1};
    float y[]={0,0,0,3,1};
    for(int i=0;i<n;i++)
    {
        PointSet[i].x=x[i];
        PointSet[i].y=y[i];
    }
    Graham_scan(PointSet,ch,n,len);
    for(int i=0;i<len;i++)
    cout<<ch[i].x<<" "<<ch[i].y<<endl;
    return 0;
}
```