发言稿

计算几何学是研究几何问题的算法，在现代工程学与数学，诸如计算机图形学、计算机辅助设计、机器人学都要应用计算几何学，在信息学竞赛中几何题也开始出现了，但是在实际的竞赛中，几何题得分率往往是最低的，所以我对几何题的算法进行了一下探索。

任何复杂的算法都是由许多简单的算法组合而成的，计算几何题也同样如此，先来看最基本的算法：

1、求直线的斜率

2、求 2 条直线的交点

3、判断 2 条线段是否相交

4、求叉积等等。

这些都是最基本的算法，是解几何题的基础，任何对基本算法的不熟悉，都可能导致解题的失败，所以熟悉几何题中的基本算法是非常重要的。

但是有了基本算法是远远不够的，因为光靠竞赛时的临时思考，组合算法从时间上来说是来不及的，这就需要熟悉一些经典算法，在竞赛中直接使用，比如：

1、求凸包

2、求最近点对

3、判断点是否在多边形内等等

基本算法和经典算法都是比较简单的，最后我们再来说一下几何题的题型及解几何题的一些技巧，

几何题的几种类型

1、 纯粹的计算求解题

解这一类题除了需要有扎实的解析几何的基础，还要全面地看待问题，仔细地分析题目中的特殊情况，比如求直线的斜率时，直线的斜率为无穷大，求 2 条直线的交点时，2 直线平行，等等。这些都是要靠平时学习时的积累。

2、 存在性问题

这一类问题可以用计算的方法来直接求解，如果求得了可行解，则说明是存在的，否则就是不存在的，但是模型的效率同模型的抽象化程度有关，模型的抽象化程度越高，它的效率也就越高，几何模型的的抽象化程度是非常低的，而且存在性问题一般在一个测试点上有好几组测试数据，几何模型的效率显然是远远不能满足要求的，这就需要对几何模型进行一定的变换，转换成高效率的模型，下面就通过一个例子来对这种方法进行一下阐述。

3、 求几何中的最佳值问题

这类问题是几何题中比较难的问题，一般没有什么非常有效的算法能够求得最佳解，最常用的是用近似算法去逼近最佳解，近似算法的优劣也完全取决于得出的解与最优解的近似程度。

[例 1]游戏者 B 在一张 100*100 纸上确定了一个目标点，游戏者 A 一开始在点(0,0)上，每次游戏者 A 从一个点到另一个点，如果新的点离目标点近了，那么游戏者 B 说“Hotter”，如果新的点离目标远了，那么游戏者 B 说“Colder”，如果距离不变，那么游戏者 B 说“Same”。

输入文件包括很多行，每行包含游戏者 A 这一步到达的点(x,y)和游戏者 B 说的话，对每次游戏者 B 说话，判断目标点可能的位置的面积，精确到小数点后 2 位。

这是一道纯粹的计算求解题，首先证明可能的位置的图形一定是个凸多边形。

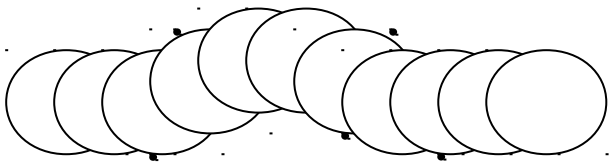
因为每次对游戏者 B 的回答，就可以确定可能的位置在出发点和到达点中垂线的哪一边或就是中垂线，每次的可能图形都是凸多边形。所以这个图形是许多个凸多边形的交集，所以这个图形是凸多边形。

接下来就是解题了，先令多边形为一个四边形，(0,0),(0,100),(100,100),(100,0)，然后对每次游戏者 B 的回答，用这条中垂线将多边形分成 2 部分，取可能的那部分，即可。

不过这样并不是完全正确的，必须考虑到特殊情况，比如游戏者 A 到达的点和这步前的点完全相同，这时就不存在中垂线了，这些都是解题中要注意的重点。

在这道题中就用到了很多解析集合的知识，包括求线段之间的交点，判断点是否在线段的两边，证明最终图形是凸多边形等等。

[例 2]在一个无限长的条形路上，有 n(n<=200)个柱子，体积不计，有一个人想从左边走到右边，人近似看成一个半径为 R 的圆（如右图），问能否实现。



拿到这道题最基本的做法是对从最左边的柱子到最右边的柱子中，每一个竖列进行扫描，计算可走到的范围，如果到最右边的柱子所在的列都有可走到的范围，则有解，否则无解。可是如果最左和最右的 2 个柱子相距非常远，那么这样计算的时间复杂度无疑是非常高的，所以我们应该对这个几何模型进行转化。

首先在这个图形中，不动的是柱子（近似看成点），动的是人（近似看成一个圆），这样处理比较麻烦，所以我们应该先把动的转换成点，圆转换成圆心是最容易想到的，对圆心来说，和柱子的距离不能<R，所以可以把每个柱子转换为以其为圆心，半径为 R 的圆，人转换成他的圆心，这样就使得计算可走到范围容易多了。

不过转换成这个模型后，问题还没有得到根本的解决，必须进一步的转换。

前 2 个算法有一个公共的特点，就是计算的都是圆外的部分，而计算圆内的部分的连通性显然比计算圆外部分来得简单，所以我们现在的目标就是把圆外部分换成圆内部分。

因为左右方向上是无穷长的，所以如果左右部分在圆外相通的话，那么上下两条直线在圆内部分就是不相通的，反之如果左右部分在圆外不相通的话，那么上下两条直线在圆内部分就是相通的，

所以我们可以将对每一个竖列的扫描转换成对每一个横行的扫描，而且又是在圆内操作，效率大大提高了。

但是前面的转换，对模型的抽象化程度却一点也没有改进，如何在这方面进行改进无疑是最关键的。

分析圆的特性，任意 2 个属于同一个圆的点必定是相通的，这就启发我们利用圆的特性，把难以处理的区域转换成几个具有代表性的点，使得能够完全表示出区域连通的特性来，到了这里，应该很容易就可以看出，取每个圆的圆心是最好不过了，因为每个圆的大小完全相同，不存在包含，相切（如果内切，就是重合了，如果外切，就是中间不连通的）等等复杂的关系，只有相交和相离的关系，而且如果 2 个圆之间相交的话，那么这 2 个圆就是相通的，可以在这 2 个圆的圆心之间连一条边，增加一个源点，与上边有交点的圆和源点连一条边，增加一个汇点，与下边有交点的圆和汇点连一条边，这样就把一道几何题完全转换成了一道图论题，只要判断源点和汇点之间是否有路就可以了，这是一道非常经典的图论题，解法就不说了。

从上面这道题的解题过程中，可以得出这样的结论：解这一类的几何题必须充分了解几何变换，挖掘题目中隐含的线索，对题目的本质进行充分的探索，才能做好几何题。

[例 3]一个农夫在一个 x*y 的矩形田地上放牧 n 头奶牛（n<=25），它们互相之间都非常仇视，所以都希望能够离其它奶牛尽量的远，它们有它们自己的标准，就是离其它奶牛的距离的倒数和越小越好，因为农夫想让它们尽量高兴，所以他必须找到一种使所有的奶牛之间的距离的倒数和尽量小，不过学历不高的农夫觉得自己很难做到，请你来为他找到这种方案，他将按照方案的解和最优解的差距来决定付给你的酬劳的多少（他知道最优解还叫你求什么呢？；））

输入 x,y,n，输出你的解的 n 头奶牛的位置。

显然这是一道求几何最值的问题，而且显然是应该用近似算法来做，那么决定解决问题质量的就是近似算法的优劣。

最简单的想法就是：既然 n 最多也只不过是 25，就来个“没有功劳，也有苦劳”吧，对在正方形上手算得到的比较好的解，按照比例放到矩形里去，比如 n=4 时，正方形上的解是四个角，而一般（x,y 之间相差不大）的矩形，这也就是最优解了。

但是这种算法在 x,y 之间差距比较大的时候，就充分显示出它的不足了，和最优解的差距非常大，几乎 1 分也得不到了，所以这种算法并不是一种非常好的方法，只能够混混而已。

在求最优解的题目中贪心法是很难有用武之地，但是贪心法是一种非常常用的近似算法，所以解这道题目，可以用贪心法一试。

第一个点取(0,0)，以后每个点都取可以使当前的倒数和最大的点，当然这里要用到逐步求精法，这样可以得到一个比较接近最优解的方案，经过分析，也很难找到可以使贪心法得到离最优解较远的数据，所以这道题中贪心法完全是一种有效的算法。

不过贪心法终究和最优解有一定距离，并不能得到所有的分数，所以需要找到另一种更加有效的算法，由于这道题没有固定的最佳解法，所以，对任何固定算法应该都有使其得不到满分的数据，所以我们又想到了非固定算法：随机化算法。

这个算法在这里我就不多介绍了，留给大家自己去思考：）

附录：程序题解

第一题:

```
program fat;
  const inputname='fat.in';
        outputname='fat.out';
  var f1,f2:text;
      x,y:array[1..100]of real;
      b:array[0..101,0..101]of boolean;
      i,j,k,n:integer;
      l,r:real;

  function dis(x1,y1,x2,y2:real):real;{求 2 点间距离的平方}

    begin
      dis:=sqr(x1-x2)+sqr(y1-y2);
    end;
  begin
    assign(f1,inputname);
    reset(f1);
    assign(f2,outputname);
    rewrite(f2);

    readln(f1,l,r);{读入输入数据}

    readln(f1,n);
    for i:=1 to n do
      readln(f1,x[i],y[i]);

    fillchar(b,sizeof(b),false);{转换成图论模型}

    for i:=1 to n do
      begin
        if y[i]<=r then
          begin
            b[0,i]:=true;
            b[i,0]:=true;
          end;
        if y[i]>=l-r then
          begin
            b[i,n+1]:=true;
            b[n+1,i]:=true;
          end;
      end;
    r:=4*r*r;
    for i:=1 to n do
      for j:=i+1 to n do
        if dis(x[i],y[i],x[j],y[j])<=r then
```

```
        begin
          b[i,j]:=true;
          b[j,i]:=true;
        end;
    for k:=0 to n+1 do
      for i:=0 to n+1 do
        for j:=0 to n+1 do
          if b[i,k] and b[k,j] then b[i,j]:=true;{求传递闭包}
    if b[0,n+1] then writeln(f2,'Not possible')
      else writeln(f2,'Possible');
    close(f1);
    close(f2);
  end.
```

第二题：

```
program game;
  const inputname='game.in';
        outputname='game.out';
  var f1,f2:text;
      i,j,t,f:integer;
      a:array[1..100,1..2]of real;
      d:array[1..100]of integer;
      xp,yp,x,y,xi1,xi2,xi3,zhi,x3,y3,x4,y4,sum:real;
      s:string;
  function sq(x1,y1,x2,y2,x3,y3:real):real;{求以(x1,y1),(x2,y2),(x3,y3)为
顶点的三角形的面积}
    begin
      sq:=abs(x1*y2+x2*y3+x3*y1-x1*y3-x2*y1-x3*y2)/2;
    end;
  begin
    assign(f1,inputname);
    reset(f1);
    assign(f2,outputname);
    rewrite(f2);
    xp:=0;yp:=0;t:=4;{初始化多边形}
    a[1,1]:=0;a[1,2]:=0;
    a[2,1]:=0;a[2,2]:=100;
    a[3,1]:=100;a[3,2]:=100;
    a[4,1]:=100;a[4,2]:=0;
    repeat
```

```
read(f1,x,y);
readln(f1,s);
while s[1]=' ' do delete(s,1,1);
if s='Same' then
  begin
    t:=0;
    continue;
  end;
 if (x=xp)and(y=yp) then continue;{考虑特殊情况：游戏者 A 没有移动}
i:=0;
xi1:=2*(x-xp);
xi2:=2*(y-yp);
xi3:=xp*xp+yp*yp-x*x-y*y;{求出中垂线的方程}
if xi1*x+xi2*y+xi3>0 then f:=1
  else f:=-1;
if s='Colder' then f:=-f;
for i:=1 to t do{判断每个点在中垂线的哪一侧}
  begin
    zhi:=xi1*a[i,1]+xi2*a[i,2]+xi3;
    if zhi>0 then d[i]:=1
      else if zhi<0 then d[i]:=-1
        else d[i]:=0;
  end;
a[t+1]:=a[1];d[t+1]:=d[1];
i:=1;
while i<=t do{增加多边形和中垂线的交点}
  begin
    if d[i]*d[i+1]<0 then
      begin
        for j:=t+1 downto i+1 do
          begin
            a[j+1]:=a[j];
            d[j+1]:=d[j];
          end;
        x3:=a[i,1];y3:=a[i,2];
        x4:=a[i+2,1];y4:=a[i+2,2];
        if x3=x4 then
          begin
```

```
              a[i+1,1]:=x3;
              a[i+1,2]:=-(xi3+xi1*a[i+1,1])/xi2;
            end
            else begin
                    a[i+1,1]:=(x3*xi2*(y4-y3)-y3*xi2*(x4-x3)-xi3*(x4-x3))/
(xi1*(x4-x3)+xi2*(y4-y3));
              a[i+1,2]:=(y4-y3)/(x4-x3)*(a[i+1,1]-x3)+y3;
            end;
          d[i+1]:=0;
          inc(t);
        end;
      inc(i);
    end;
  i:=1;
  while i<=t do{删除不符合要求的点}
    begin
      if d[i]=-f then
        begin
          for j:=i+1 to t do
            begin
              a[j-1]:=a[j];
              d[j-1]:=d[j];
            end;
          dec(t);
        end
        else inc(i);
    end;
  sum:=0;
  for i:=2 to t-1 do{求出多边形的面积}
    begin
      sum:=sum+sq(a[1,1],a[1,2],a[i,1],a[i,2],a[i+1,1],a[i+1,2]);
    end;
  writeln(f2,sum:0:2);
  xp:=x;yp:=y;
 until eof(f1);
 close(f1);
 close(f2);
end.
```

第三题:贪心法

```
program enemy;
 const inputname='enemy.in';
```

```
      outputname='enemy.out';
var f1,f2:text;
    d:array[1..25,1..2]of real;
    i,j,k,l,x,y,n,bj,bk:integer;
    x1,y1,r,x2,y2,bs,sum,xz,yz:real;
    b1:boolean;
function dis(x1,y1,x2,y2:real):real;{求(x1,y1),(x2,y2)间的距离}
  begin
    dis:=sqrt(sqr(x1-x2)+sqr(y1-y2));
  end;
begin
  assign(f1,inputname);
  reset(f1);
  assign(f2,outputname);
  rewrite(f2);
  readln(f1,x,y,n);
  d[1,1]:=0;
  d[1,2]:=0;
  if n>=2 then
    begin d[2,1]:=x;d[2,2]:=y;end;
  for i:=3 to n do{依次贪心 n 头奶牛的位置,使当前倒数和最小}
    begin
      x1:=x/2;y1:=y/2;r:=100;
      xz:=x/4;yz:=y/4;
      while (xz>0.002)or(yz>=0.002) do{对整个田地进行分割求解，找
寻当前最佳位置}
        begin
          bs:=1000000;
          bj:=0;bk:=0;
          for j:=-2 to 2 do
            begin
              x2:=x1+j*xz;
              if (x2>x)or(x2<0) then continue;
              for k:=-2 to 2 do
                begin
                  y2:=y1+k*yz;
                  if (y2>y)or(y2<0) then continue;
                  b1:=true;
                  for l:=1 to i-1 do
                    if (x2=d[l,1])and(y2=d[l,2]) then
```

```
                begin
                  b1:=false;
                  break;
                end;
              if not b1 then continue;
              sum:=0;
              for l:=1 to i-1 do
                sum:=sum+1/dis(d[l,1],d[l,2],x2,y2);
              if sum<bs then
                begin
                  bs:=sum;
                  bj:=j;bk:=k;
                end;
            end;
          end;
        x1:=x1+bj*xz;
        y1:=y1+bk*yz;
        xz:=xz/4;yz:=yz/4;
      end;
    d[i,1]:=x1;d[i,2]:=y1;
  end;
 for i:=1 to n do
  writeln(f2,d[i,1]:0:2,' ',d[i,2]:0:2);
 close(f1);
 close(f2);
end.
```