信息科学系
低年级讨论班

（4）

裘宗燕

北京大学数学学院信息科学系

2010年3月～6月

为“The Book”写程序

《代码之美》第33章

“The Book”

- **Paul Erdos (例如：http://www.douban.com/group/topic/3017467/)** 常说到**“The Book”**（或许可称为“天书”），里面包含着所有数学定理的最优证明（ **Paul Erdos**，一位与许多人合作的当代数学家）
- 本文作者提出“应该”也有一本算法和程序的**“The Book”**，里面包含所有可计算问题的最优解决方案（及实现算法的最优程序）
- 作者的问题：我们能不能为这本书写一个程序？

  也就是说，针对一个问题写出一个不仅无懈可击，而且每个读者都只能赞不绝口、高山仰止的程序
- 人都希望自己能创造出这样的算法艺术瑰宝，但

  程序里常会有些瑕疵，有不如意之处

  即使正确，也常有勉强和不自然之处、结构脆弱之处、逻辑较混乱的特殊情况和例外情况
- 在这些方面努力，可能突然获得灵感，或通过别人工作的启发，得到一个绝妙的可以写入**“The Book”**的程序

本文的问题

- 作者讲了自己的一个故事：不断追求最终写出了一个好程序
- 问题领域：计算几何。这里的情况和特点：
  - 许多问题看似容易，深入之后会发现很麻烦
  - 计算集合中的算法不是以象素为单位（不像图形学），而是理想化的（像尺规作图）：点无大小，线宽为0，圆是精确的，等等
  - 必须得到精确结果，一点不精确都可能改变整个计算的性质。例如小数点后最低位的数字也可能逆转结果的性质
- 欧几里得说：几何无捷径。现在还加上效率。这里主要关心美学
- 问题：写一个函数，判断平面三点是否共线
  - 似乎很简单，有许多数学上正确的方法
  - 肯定有许多人做过
  - 可以找别人的做法，但作者是自己考虑

Lisp

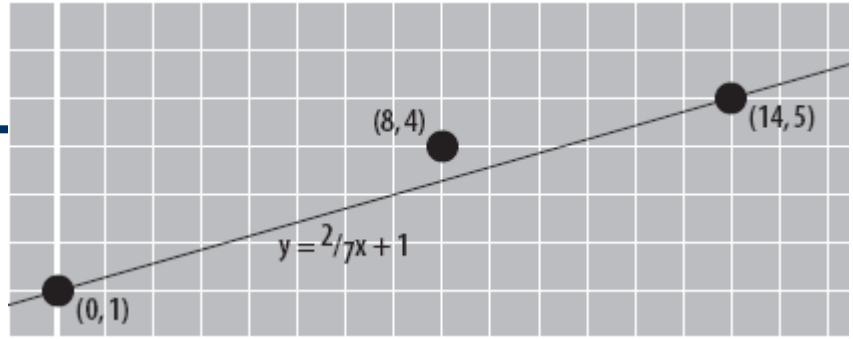
- 下面要给出函数的几个版本，都是用 **Lisp** 写的
- 例：求最大公约数的欧几里得算法

```
(defun gcd (a b)
   (if (= b 0)
      a
      (gcd b (mod a b))))
```

这个程序肯定在**“The Book”**里面

- **Lisp** 采用括号括起的前缀表达式，括号里
  - 第一个表达式表示操作，后面是操作对象
  - **defun** 定义函数，随后是函数名，参数表和函数体
  - **if** 是条件表达式，后面三项分别是条件和两个分支

朴素算法（第一试）



- 用纸和笔计算（若目测不能确定）：
  - 在纸上画出三个点
  - 过两点画直线，看第三个点是否在线上
- 计算机上也可以这样做
  - 需要选两个点画直线，怎样选择最好？为什么？
  - 请大家讨论
- 假设直线方程为 **y = mx + b**，假定三个点是 **p, q, r**，用两点算方程，用第三点坐标判断是否在线上，函数定义如下：

```
(defun naive-collinear (px py qx qy rx ry)
  (let ((m (slope px py qx qy))
        (b (y-intercept px py qx qy)))
    (= ry (+ (* m rx) b))))
```

let 算出斜率和截距，最后带入 **rx** 看算出的值是否等于 **ry**

问题（第一试）

- 这个函数在否正确？
  - 通常没问题，用大量随机数据试验，也未必会发现问题
- 函数有错。例：**(0 0)，(0 1)，(0 2)** 显然共线

  但前面函数不能给出正确结果，原因在斜率计算：

```
(defun slope (px py qx qy)
   (/ (- qy py) (- qx px))))
```

  当 **qx = px** 时出现除**0**错误，函数失效
- 想想在 **C** 语言里怎么写这个函数，怎么表现其出错情况？
- 在 **Lisp** 里可以用特殊值 **nil** 表示特殊情况

```
(defun slope (px py qx qy)
  (if (= px qx)
      nil
      (/ (- qy py) (- qx px))))
```

处理特殊情况（第一试，修正）

- 截距也有类似问题：如果两点 **qy = py**。类似处理

```
(defun y-intercept (px py qx qy)
  (let ((m (slope px py qx qy)))
    (if (not m)      ; m 为 nil 时 (not m) 是真
        nil
        (- py (* m px)))))
```

- 判断三点共线的函数里也需要处理特殊情况，新定义：

```
(defun less-naive-collinearp (px py qx qy rx ry)
  (let ((m (slope px py qx qy))
        (b (y-intercept px py qx qy)))
    (if (numberp m)
        (= ry (+ (* m rx) b))
        (= px rx))))   ; 当 qx = px 时，如果 rx = px 则三点共线
```

函数定义已经变得很不干净清晰了

分析

- 虽然解决问题只需在几个地方加一两行代码，但却给程序带来不必要的复杂性，这“说明”可能有什么地方想的不对
- 按说垂直线或水平线应该和其他的直线没有本质区别，由于现在是根据 **y** 轴计算斜率，所以就出了问题。可以考虑其他思考方式
- 一个可能性是旋转坐标系，共线关系不受坐标系旋转影响

  选择坐标系只需做矩阵乘法，代价并不高

  但还是需要做了判断之后选择，不能改善程序的结构

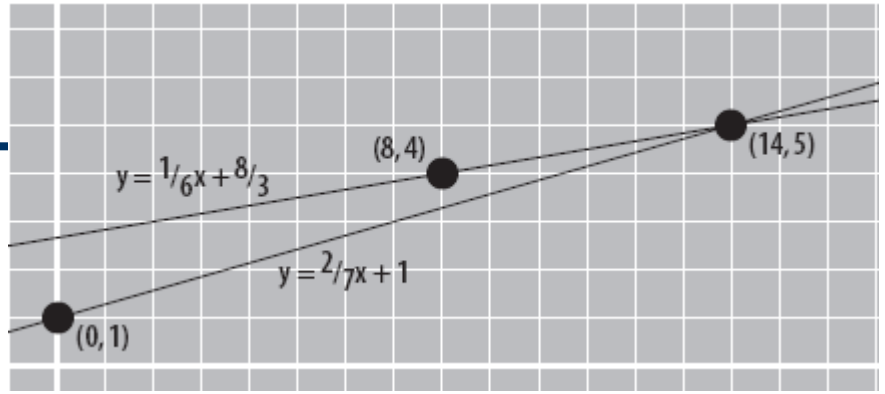
  不符合需要：找一种判断共线性质的简单办法
- 某网站推荐的方法：发现要出现除**0**，就把结果设置为 10^{10}

  从实用角度，这种策略能否在多数情况下工作得好？值得怀疑

  但如果输入来自实际，数据的误差完全可能大于 $1/10^{10}$

  总之，这种策略太草率，**“The Book”**里不会这样做

另辟蹊径（第二试）



- 方法二，通过每对点画一条线

  可只画过一点两直线后比斜率

- 这样写出的共线判断函数

```
(defun mm-collinear (px py qx qy rx ry)
  (equalp (slope px py qx qy)
          (slope qx qy rx ry))) ; 注意到什么？
```

- 注意：这里没用 **if** 而是直接比较，垂线情况也能统一处理
- 这里的简洁有欺骗性，依赖于 **Lisp** 细节：**equalp** 不仅能比较数值，两个比较对象都是 **nil** 时也返回真。如果有严格类型，函数只能写

```
(defun typed-mm-collinear (px py qx qy rx ry)
  (let ((pq-slope (slope px py qx qy))
        (qr-slope (slope qx qy rx ry)))
    (or (and (numberp pq-slope)  (numberp qr-slope)
             (= pq-slope qr-slope))
        (and (not pq-slope)   (not qr-slope)))))  ; 不那么漂亮了
```

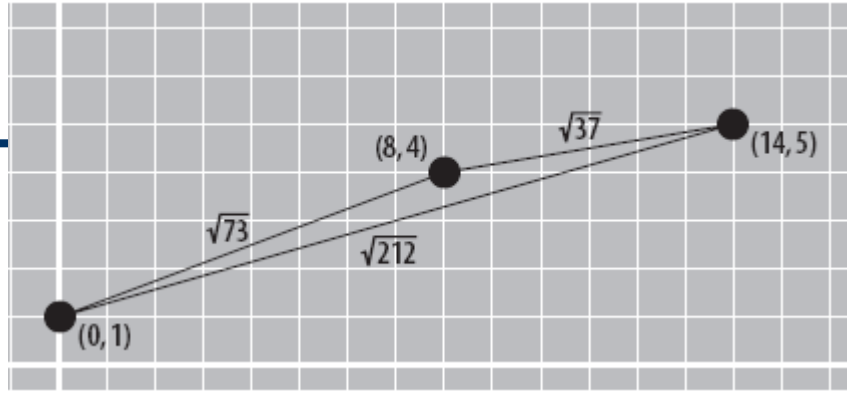
分析（斜率不可靠）

- 但这个函数在逻辑上好些：两个斜率都能算出且相同，或都不能算出，则三点共线
- 本希望将这个函数作为最后版本，但又发现了新问题：
  - 两版本都能区分共线点和几乎共线的点
  - 但对 **(0 0)，(0 0)，(1 1)** 都出了问题
- 首先：这种情况应该怎样看？

  应该看作共线情况的特例，因为有一条直线过这三个点
- 过两个相同的点可以画任意多条直线，前面函数都没考虑这种情况
  - 对 **(0 0)，(0 0)，(1 1)** 返回 **nil**
  - 对 **(0 0)，(0 0)，(0 1)** 返回 **t**

  这是明显的病态行为
- 加约束条件禁止两点相同（结果无一般性），或通过检查排除这些问题（复杂的判断逻辑），都不应该是**“The Book”**里的做法

再寻它途（三角不等式）



- 另一方法基于一个观察：

  不共线的三点确定一个三角形

  如共线，长边等于两个短边之和

- 可考虑计算三边长，而后做上述判断。写出的函数

```
(defun triangle-collinear (px py qx qy rx ry)
  (let ((pq (distance px py qx qy))
        (pr (distance px py rx ry))
        (qr (distance qx qy rx ry)))
    (let ((sidelist (sort (list pq pr qr) #'>))) ; 得到从大到小排序的表
      (= (first sidelist)
         (+ (second sidelist) (third sidelist))))))
```

- 得到清晰性，付出的代价很大：失去了计算的精确性

  计算中需要求平方根，一般不可能精确

问题（第三试）

- 求距离函数

```
(defun distance (px py qx qy)
  (sqrt (+ (square (- qx px))
           (square (- qy py)))))
```

  求平方根是产生误差的根源，可能把很低的三角形判为共线

- 这个问题没有简单处理方法
  - 可认为是真实世界的一个**bug**：有理数的点会产生无理数的距离
- 可能考虑计算机代数等等，用符号计算处理

  通过代数变换，可能两个距离确实相等
- **“The Book”**里不应该这样写

  简单的问题应该有简单的算法和简单的程序

河道弯曲模型

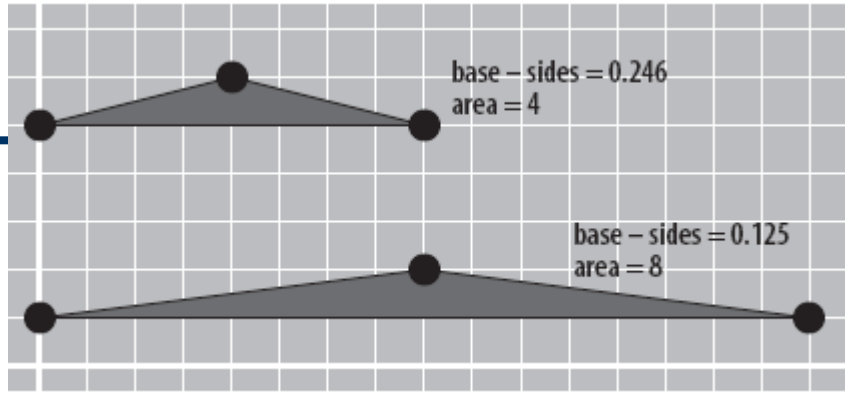
- 问题来自实际：需要为河道弯曲建立一个简单模型
  - 基本想法是把光滑的河道分解为一系列短的直线段
  - 需要基于这些线段之间的夹角度量河道的曲率
  - 特别是需要关注曲率为0的地方，因此需要判断共线的谓词
- 另一部分更麻烦：随着河道变得越来越弯曲，会出现河流冲破河道之间的阻挡而走捷径的现象（这时会留下“牛胃”湖）
  - 要在模型里检查这种事件，需要通过扫描发现线段的交点
  - 这样考虑下去，可能能编写出函数，但会出现极复杂的判断结构
  - 不仅要考虑水平线和垂直线，还可能要考虑平行线
- 其他人也考虑类似问题。如一篇关于线段交叉的文章说：

  为突出问题，先假定两个点的x和y坐标都互不相同，这样就排除了水平线和垂直线的情况……

其他人的考虑

- 作者写了一篇有关线段交叉的博客文章请求帮助。得到许多回复。如
  - 建议用极坐标
  - 提出用参数方程表示线性方程（使 **x** 和 **y** 成为 **t** 的函数）
  - 通过仿射变换，将一条边变换到 **(-1 0)** 和 **(1 0)** 之间
  - 建议不在欧几里得几何里做这件事，而到在射影平面上去做。在那里有“一个无穷点”，对解决问题有帮助
- 最后是**Jonathan Richard Shewchuk**给了作者一个链接，指向他的讲义，论文和可用的代码
- 从中的收获：
  - 有几种方法可能用于处理线段交叉问题
  - 找到了一个可以写入**“The Book”**的共线判断算法

最后结果



- 基本想法：求面积

  三点共线 **iff** 张成的三角形面积 **= 0**

- 优点：
  - 只需简单的基本运算
  - 不需要判断条件
- 从三个点坐标计算面积有许多方法
  - **½ b h**
  - 三角方法等
- 最好用的是求矢量叉积的方法（得到带符号面积）

$$A = \frac{1}{2}\begin{vmatrix} x_1 - x_3 & y_1 - y_3 \\ x_2 - x_3 & y_2 - y_3 \end{vmatrix} = \frac{1}{2}[(x_1 - x_3)(y_2 - y_3) - (x_2 - x_3)(y_1 - y_3)]$$

共线判断函数

- 去掉无关情况，也不需要求三角形面积（求平行四边形就行），进一步化简之后得到：

```
(defun area-collinear (px py qx qy rx ry)
  (= (* (- px rx) (- qy ry))
     (* (- qx rx) (- py ry))))
```

只用几个简单运算

没有条件判断、斜率/截距、平方根，不会出除0错误，等等

- 如果是处理浮点数，其特征仍然不容易看清楚。但它比基于周长或比较距离的方法好得多了
- **Shewchuk**还给出了一段判断共线的高度优化的**C**代码，其中尽可能使用浮点运算，必要时转到精确运算

总结

- 正确答案是无意中发现的（但这种追求很重要）
- 应该毫不迟疑地寻求帮助
- 找到别人的方法 **vs** 自己解决问题/发明方法
- 对于小程序，容易推倒重新来；对于大程序，则更需慎重
- 在编程或软件开发过程中，应该花多少努力去追求“完美代码”？
  - 数学家 **Hardy** 说“丑陋的数学不可能永存”
  - 计算机科学技术领域也会是这样吗？
- 换种方式问类似问题
  - 对每个严格表述的计算问题，都有可以写入**“The Book”**的程序吗？
  - **“The Book”**会不会有空白页？（对某些问题没有回答）

讨论