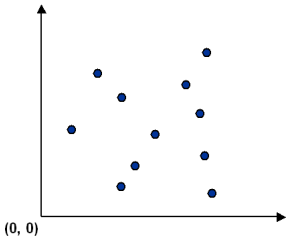
共线性点

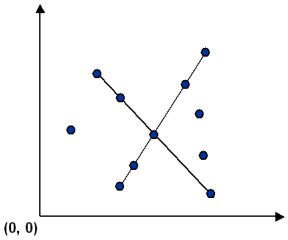
规范 常见问题 项目 提交

编写程序以识别给定点集中的线模式。

计算机视觉包括分析视觉图像中的模式并重建产生它们的真实对象。该过程通常分为两个阶段：*特征检测*和*模式识别*。特征检测涉及选择图像的重要特征;模式识别涉及发现要素中的模式。我们将调查一个特别干净的模式识别问题，涉及点和线段。这种模式识别在许多其他应用（如统计数据分析）中出现。

问题所在了给定平面中的一组n个不同点，查找连接 4 个或更多点的子集的每个（最大）线段。





点数据类型。通过实现以下 API 创建表示平面中点的不可变数据类型：`Point`

```
public class Point implements Comparable<Point> {
   public Point(int x, int y)                         // constructs the point (x, y)

   public   void draw()                               // draws this point
   public   void drawTo(Point that)                   // draws the line segment from this point to that point
   public String toString()                           // string representation

   public              int compareTo(Point that)      // compare two points by y-coordinates, breaking ties by x-coordinates
   public           double slopeTo(Point that)        // the slope between this point and that point
   public Comparator<Point> slopeOrder()              // compare two points by slopes they make with this point
}
```

要开始使用数据类型Point.java ，它实现了构造函数和 、和 方法。您的工作是添加以下组件。`draw()drawTo()toString()`

- 该方法应按y坐标比较点，按x坐标断开关系。从形式上讲，调用点 (x_0, y_0) *小于*参数点 (x_1, y_1) ， 如果并且仅当$y_0 < y_1$或如果$y_0 = y_1$和$x_0 < x_1$`.compareTo()`
- 该方法应返回调用点 (x_0, y_0) 和参数点 (x_1, y_1) 之间的斜率， 由公式 $(y_1 = y_0) / (x_1 = x_0)$ 给出的斜率。将水平线段的斜率视为正零;将垂直线段的斜率视为正无穷大;将退化线段的斜率（在点和自身之间）视为负无穷大。`slopeTo()`
- 该方法应返回一个比较器，该比较器将两个参数点与调用点 (x_0, y_0) 的斜率进行比较。从形式上讲，如果且仅当斜率 $(y_1 = y_0) / (x_1 = x_0)$ 小于斜率 $(y_2 = y_0) / (x_2 = x0)$ 时， 点 $(x2, y2)$ 小于点 $(x_2, y_2) = 0)$ 时，将水平线段、垂直段和退化线段视为方法中的方法。`slopeOrder()slopeTo()`
- *不要*重写 或 方法。`equals()hashCode()`

*角箱。*为了避免整数溢出或浮点精度的潜在复杂情况，您可以假定构造函数参数和每个参数介于 0 和 32， 767 之间。`xy`

线段数据类型。要表示平面中的线段，请使用数据类型LineSegment.java ，该数据类型具有以下 API：

```
public class LineSegment {
   public LineSegment(Point p, Point q)       // constructs the line segment between points p and q
   public   void draw()                       // draws this line segment
   public String toString()                   // string representation
}
```

蛮力。编写一个程序，一次检查 4 个点，并检查它们是否都位于同一线段上，返回所有此类线段。要检查 4 点p、q、r和s是否为共线，请检查p和q之间、p和r之间以及p和s之间的三个斜率是否均相等。 `BruteCollinearPoints.java`

```
public class BruteCollinearPoints {
   public BruteCollinearPoints(Point[] points)    // finds all line segments containing 4 points
   public           int numberOfSegments()        // the number of line segments
   public LineSegment[] segments()                // the line segments
}
```

该方法应包含每个线段，仅包含 4 个点一次。如果 4 个点出现在顺序$p=q=r=s$的线段上，则您应该包括线段$p=s$或$s=p$（但不是两者），并且不应包括*子段*，如$p=r$或$q=r$。为简单起见，我们不会向具有 5 个或更多共线点的任何输入提供。`segments()BruteCollinearPoints`

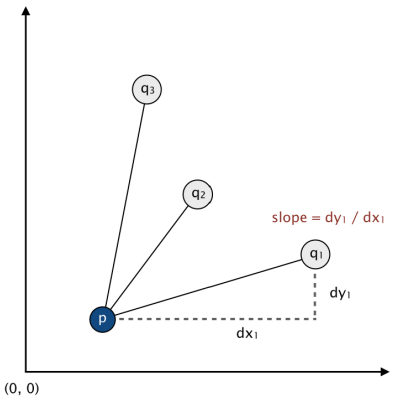
*角箱。*如果数组中的任何点为 ， 则将 参数引发 到 ， 如果 数组中的任何点为 ， 或者如果构造函数的参数包含重复点。
`IllegalArgumentExceptionnullnull`

*性能要求。*在最坏的情况下，程序运行时间的增长顺序应为n^4，并且应使用与n成比例的空间加上返回的线段数。

更快的、基于排序的解决方案。值得注意的是，与上述暴力解相比，解决问题的速度要快得多。给定点p，以下方法确定p是否参与一组 4 个或更多共线点。

- 认为p是起源。
- 对于彼此点q，确定它与p的斜率。
- 根据它们用p的坡度对点进行排序。
- 检查排序顺序中的任何 3 个（或更多）相邻点相对于p具有相等的坡度。如果是这样，这些点以及p是共线性的。

将此方法应用于每个n个点，从而产生一种有效的算法来解决问题。该算法解决了这个问题，因为相对于p具有相等坡度的点是共线性的，排序将这些点合并在一起。该算法速度快，因为瓶颈操作是排序。



编写一个实现此算法的程序。`FastCollinearPoints.java`

```
public class FastCollinearPoints {
   public FastCollinearPoints(Point[] points)     // finds all line segments containing 4 or more points
   public           int numberOfSegments()        // the number of line segments
   public LineSegment[] segments()                // the line segments
}
```

该方法应包含每个*最大*线段，仅包含 4 个（或更多）点一次。例如，如果 5 个点出现在线段上，顺序$p=q=r=s=t$，则不包括子段$p=s$或$q=t$。
`segments()`

*角箱。*如果数组中的任何点为 ， 则将 参数引发 到 ， 如果 数组中的任何点为 ， 或者如果构造函数的参数包含重复点。
`IllegalArgumentExceptionnullnull`

*性能要求。*在最坏的情况下，程序运行时间的增长顺序应为$n^2 \log n$，并且应使用与n成比例的空间加上返回的线段数。 即使输入具有 5 个或更多共线点，也应正常工作。`FastCollinearPoints`

示例客户端。此客户端程序将输入文件的名称作为命令行参数;读取输入文件（以下面指定的格式）;打印到程序发现的行段的标准输出，每行一个;并绘制到标准绘制线段。

```
public static void main(String[] args) {

    // read the n points from a file
    In in = new In(args[0]);
    int n = in.readInt();
    Point[] points = new Point[n];
    for (int i = 0; i < n; i++) {
        int x = in.readInt();
        int y = in.readInt();
        points[i] = new Point(x, y);
    }

    // draw the points
    StdDraw.enableDoubleBuffering();
    StdDraw.setXscale(0, 32768);
    StdDraw.setYscale(0, 32768);
    for (Point p : points) {
        p.draw();
    }
    StdDraw.show();

    // print and draw the line segments
    FastCollinearPoints collinear = new FastCollinearPoints(points);
    for (LineSegment segment : collinear.segments()) {
        StdOut.println(segment);
        segment.draw();
    }
    StdDraw.show();
}
```

输入格式。我们以以下格式提供几个示例输入文件（适合与上述测试客户端一起使用）：一个整数n，然后是n对整数(x, y)，每个在 0 和 32,767 之间。下面是两个示例。

```
% cat input6.txt
6
19000  10000
18000  10000
32000  10000
21000  10000
 1234   5678
14000  10000
```

```
% cat input8.txt
8
 10000      0
     0  10000
  3000   7000
  7000   3000
 20000  21000
  3000   4000
 14000  15000
  6000   7000
```

```
% java-algs4 BruteCollinearPoints input8.txt
(10000, 0) -> (0, 10000)
(3000, 4000) -> (20000, 21000)

% java-algs4 FastCollinearPoints input8.txt
(3000, 4000) -> (20000, 21000)
(0, 10000) -> (10000, 0)

% java-algs4 FastCollinearPoints input6.txt
(14000, 10000) -> (32000, 10000)
```

Web 提交。提交仅包含 、和 的 .zip 文件。我们将供应 和 。除了 、和 中的函数外，不得调用任何库函数。仅当库中已引入库函数时，才能使用库函数。例如，可以使用 ，但不能使用 。

BruteCollinearPoints.javaFastCollinearPoints.javaPoint.javaLineSegment.javaalgs4.jarjava.langjava.utilalgs4.jarjava.utilArrays.sort()java.util.Has