# Физико-математический лицей № 239

Поиск пары точек, дистанция которой будет минимальна

# Отчет о годовом проекте по информатике

Работу выполнила Ученица 10-7 класса Галяутдинова Динара

Санкт-Петербург

2021

## Постановка задачи

Дано множество точек на плоскости. Выберем из этого множества две точки и проведем через них прямую. Назовем дистанцией такую максимальную величину, что на расстоянии от прямой не меньше, чем дистанция лежит хотя бы половина оставшихся точек множества (кроме этих двух). Найти такую пару точек, у которой дистанция будет максимальна. В качестве ответа: выделить эти две точки, нарисовать проходящую через них прямую, выделить точки, лежащие на дистанции, нарисовать дистанцию (отрезок от одной из самых удаленных точек до прямой), а также "коридор" (две прямые, параллельные найденной прямой, находящиеся на найденной дистанции).

## Уточнение исходных и выходных данных и ограничений на них

### Исходные данные

* Координаты точек из вводимого нами множества точек A*.*
* Координаты двух выбранных из множества точек.

### Выходные данные

На экран необходимо вывести и показать:

* Дистанцию — отрезок от одной из самых удаленных точек до прямой.
* Координаты пары точек, дистанция которой будет минимальна; проходящая через них прямая.
* Точки, лежащие на дистанции;
* "коридор" — две прямые, параллельные найденной прямой, находящиеся на найденной дистанции.

## Выбор метода решения

### Анализ исходных данных и выбор используемой структуры данных

В задаче не требуется использовать информацию о конкретном расстоянии от конкретной точки до прямой. Анализируется расстояния от каждой точки до прямой. За один проход (одну линейную операцию чтения) по исходным данным можно накапливать расстояния от точек до прямой. Для этого достаточно двух массивов - из данных точек множества и величин расстояний – (из типа points и double соответственно).

### Выбор метода решения

Считываем построчно данные из файла по каждой точке. Строим прямую, проходящую через две выбранные точки (класс Line). Прямую можно задавать координатами двух точек или коэффициентами A, B, C уравнения прямой. Рассчитываем расстояние от каждой точки до прямой с помощью метода *distanceToPoint*. Заполняем полученными расстояниями динамический массив *tDist* .

Сравниваем (упорядочиваем) получившиеся расстояния в динамическом массиве *tDist* и выбраем [n/2 – 0,1] значение длины как дистанцию, где n - количество точек множества без двух выбранных точек. Таким образом, мы находим такую пару точек, у которой дистанция будет максимальна.

Выделяем точки, лежащие на дистанции (точки, расстояние до которых равно дистанции).

Выделяем точки, лежащие внутри дистанции (точки, расстояние до которых меньше дистанции).

Рисуем «коридор» — две прямые, параллельные построенной прямой, находящиеся на найденной дистанции (строим две прямые, параллельные прямой, проходящей через две выбранные точки, и находящие от неё на расстоянии, равном дистанции) решая простую геометрическую задачу, заполняем динамический массив *getRanges*, хранящий координаты точек, которые будут принадлежать требуемым прямым. Для построения двух прямых также использовался метод *oneSide*, определяющий, лежат ли точки по одну сторону от данной прямой.

Задача решена.

## Составление алгоритма

### Обобщенная блок-схема алгоритма

Начало

|  |  |  |  |
| --- | --- | --- | --- |
|  | |  | |
|  | Чтение исходных данных. Сбор статистики. Выделить две выбранные точки, нарисовать прямую проходящую через них | |  |
|  | |  | |
|  | Вычисление расстояния от каждой точки до прямой. Заполнение динамического массива *tDist* значениями расстояний от точек до прямой | |  |
|  | |  | |
|  | Сортировка списка *tDist* <==> Нахождение такой пары точек, у которой дистанция будет максимальна. Выделить точки, лежащие на дистанции (точки, расстояние до которых равно дистанции). Выделяем точки, лежащие внутри дистанции (точки, расстояние до которых меньше дистанции): | |  |
|  | |  | |
|  | Рисуем «коридор» — две прямые, параллельные найденной прямой, находящиеся на найденной дистанции: заполняем динамический массив *getRanges*, хранящий координаты точек, которые будут принадлежать требуемым прямым, решая простую геометрическую задачу, при этом определять, лежат ли точки по одну сторону от данной прямой с помощью метода *oneSide.* | |  |
|  | |  | |

## Листинг программы

## Класс problem :

## package problem;

|  |
| --- |
|  |
|  |  |
|  | import javax.media.opengl.GL2; |
|  | import java.io.\*; |
|  | import java.util.\*; |
|  |  |
|  | **// Класс задачи** |
|  | public class Problem { |
|  |  |
|  | **// текст задачи и заголовок окна** |
|  |  |
|  | public static final String PROBLEM\_TEXT = "ПОСТАНОВКА ЗАДАЧИ:\n" + |
|  | "Дано множество точек на плоскости. Выберем из этого множества две точки и проведем через них прямую.\n" + |
|  | "Назовем дистанцией такую максимальную величину, что на расстоянии \n" + |
|  | "от прямой не меньше, чем дистанция лежит хотя бы половина оставшихся точек \n" + |
|  | "множества (кроме этих двух). Найти такую пару точек, у которой дистанция будет минимальна.\n" + |
|  | " В качестве ответа: выделить эти две точки, нарисовать проходящую через них прямую,выделить \n" + |
|  | "точки, лежащие на дистанции, нарисовать дистанцию (отрезок от одной из самых удаленных\n" + |
|  | "точек до прямой), а также <<коридор>> (две прямые, параллельные \n" |
|  | + "найденной прямой, находящиеся на найденной дистанции)"; |
|  | public static final String PROBLEM\_CAPTION = "Итоговый проект ученицы 10-7 Галяутдиновой Динары"; |
|  |  |
|  |  |
|  | **// путь к файлу** |
|  |  |
|  | private static final String FILE\_NAME = "points.txt"; |
|  |  |
|  |  |
|  | **// список точек** |
|  |  |
|  | public ArrayList<Point> points; |
|  | public ArrayList<Point> resPoints; |
|  |  |
|  | public Point resA = null; |
|  | public Point resB = null; |
|  | Line answline = null; |
|  | Line rangeALine = null; |
|  | Line rangeBLine = null; |
|  | double minDist; |
|  |  |
|  |  |
|  | **// Конструктор класса задачи** |
|  |  |
|  | public Problem() { |
|  | points = new ArrayList<>(); |
|  | resPoints = new ArrayList<>(); |
|  | minDist = 1e10; |
|  | } |
|  |  |
|  |  |
|  | **// Добавление точки** |
|  |  |
|  | public void addPoint(double x, double y) { |
|  | Point point = new Point(x, y); |
|  | points.add(point); |
|  | } |
|  |  |
|  | public void addRandomPoints(int n) { |
|  | for (int i = 0; i < n; i++) {  Point p = Point.getRandomPoint(); |
|  |  |
|  | points.add(p); |
|  | }  } |
|  |  |
|  | public void clear() { |
|  | points.clear();  resPoints.clear(); |
|  | public void clear() { |
|  | resA = null;  resB = null;  answline = null; |
|  |
|  | minDist = 1e10; |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | public void solve() { |
|  |
|  |  |
|  | **//выделение точек, дистанция которых меньше дистанции:** |
|  |  |
|  | for (Point point : points) { |
|  | if (answline.distanceToPoint(point) < minDist)  resPoints.add(point); |
|  |  |
|  | } |
|  |  |
|  | **// Загрузка задачи из файла** |
|  |  |
|  | public void loadFromFile() {  clear();  try {  File file = new File(FILE\_NAME);  Scanner sc = new Scanner(file); |
|  |
|  |  |
|  | while (sc.hasNextLine()) {  double x = sc.nextDouble();  double y = sc.nextDouble();  sc.nextLine(); |
|  | Point point = new Point(x, y); |
|  | points.add(point);  }  catch (Exception ex) {  System.out.println("Ошибка чтения из файла: " + ex);  }  } |
|  |  |
|  | **//** **Сохранить задачу в файл** |
|  |  |
|  | public void saveToFile() { |
|  | try {  PrintWriter out = new PrintWriter(new FileWriter(FILE\_NAME));  for (Point point : points) {  out.printf("%.2f %.2f\n", point.x, point.y);  }  out.close();  } catch (IOException ex) {  System.out.println("Ошибка записи в файл: " + ex);  }  } |
|  |  |
|  |  |
|  |  |
|  |
|  |

**// Рисовка**

public void render(GL2 gl) {

gl.glColor3d(1, 1, 1);

for (Point point : points) {

point.render(gl);

}

gl.glColor3d(1, 0.2, 0.7);

if (answline != null && resA != null && resB != null) {

answline.renderLine(gl);

Figure.renderPoint(gl, resA.x, resA.y, 7);

Figure.renderPoint(gl, resB.x, resB.y, 7);

rangeALine.renderLine(gl);

rangeBLine.renderLine(gl);

gl.glColor3d(0, 1, 0);

for (Point point : answline.getPoints(minDist)) {

Figure.renderPoint(gl, point.x, point.y, 5);

}

}

gl.glColor3d(0, 0, 1);

for (Point point : resPoints) {

point.render(gl);

}

}

}

**Класс Line :**

**// Конструктор**

public class Line {

private double a;

private double b;

private double c;

double x1;

double y1;

double x2;

double y2;

public Line(double x1, double y1, double x2, double y2) {

this.a = y1 - y2;

this.b = x2 - x1;

this.c = x1 \* y2 - x2 \* y1;

this.x1 = x1;

this.y1 = y1;

this.x2 = x2;

this.y2 = y2;

}

public Line(double a, double b, double c) {

this.a = a;

this.b = b;

this.c = c;

this.x1 = x1;

this.y1 = y1;

this.x2 = x2;

this.y2 = y2;

}

**// Получение случайной точки**

static Line getRandomLine() {

Random r = new Random();

double x1 = r.nextDouble() \* 2 - 1;

double y1 = r.nextDouble() \* 2 - 1;

double x2 = r.nextDouble() \* 2 - 1;

double y2 = r.nextDouble() \* 2 - 1;

return new Line(x1, y1, x2, y2);

}

**//рисование прямой**

public void renderLine(GL2 gl) {

gl.glLineWidth(3);

gl.glBegin(GL.GL\_LINES);

int samples = 100;

double dx = x2 - x1;

double dy = y2 - y1;

double l = Math.sqrt(dx \* dx + dy \* dy);

double ndx = dx / l \* 10;

double ndy = dy / l \* 10;

gl.glVertex2d(x1 + ndx, y1 + ndy);

gl.glVertex2d(x1 - ndx, y1 - ndy);

gl.glEnd();//end drawing of points

}

**//нахождение расстояния от точки до прямой**

public double distanceToPoint(Point p) {

double f = Math.sqrt(a \* a + b \* b);

double k = a \* p.x + b \* p.y + c;

double q = Math.abs(k / f);

return q; //расст от точки p до прямой line

}

**//определение, по одну ли сторону от прямой находятся точки**

public boolean oneSide(Point p1, Point p2) {

if (((a \* p1.x + b \* p1.y + c <= 0.001) && (a \* p2.x + b \* p2.y + c <= 0.001)) || ((a \* p1.x + b \* p1.y + c >= 0.001) && (a \* p2.x + b \* p2.y + c >= 0.001))) {

return true;

} else {

return false;

}

}

**//динамический маасив, хранящий длины расстояний**

public ArrayList<Line> getRanges(double dist) {

double xo = 0.5;

double yo = (a \* xo + c) / (-b);

double cl = dist / Math.cos(Math.atan2(-a, b));

double cx = dist / Math.sin(Math.atan2(-a, b));

Point p1l = new Point(xo, yo + cl);

Point p2l = new Point(xo - cx, yo);

Point p1r = new Point(xo + cx, yo);

Point p2r = new Point(xo, yo - cl);

ArrayList<Line> ranges = new ArrayList<>();

if (oneSide(p1l, p2l)) {

ranges.add(new Line(xo, cl + yo, xo - cx, yo));

}

if (oneSide(p1l, p1r)) {

ranges.add(new Line(xo, cl + yo, xo + cx, yo));

}

if (oneSide(p2l, p1r)) {

ranges.add(new Line(xo - cx, yo, xo + cx, yo));

}

if (oneSide(p2l, p2r)) {

ranges.add(new Line(xo - cx, yo, xo, yo - cl));

}

if (oneSide(p1r, p2r)) {

ranges.add(new Line(xo + cx, yo, xo, yo - cl));

}

if (oneSide(p1l, p2r)) {

ranges.add(new Line(xo, cl + yo, xo, yo - cl));

}

return ranges;

}

**//динамический маасив, хранящий искомые точки**

public ArrayList<Point> getPoints(double dist) {

double xo = 0.5;

double yo = (a \* xo + c) / (-b);

double cl = dist / Math.cos(Math.atan2(-a, b));

double cx = dist / Math.sin(Math.atan2(-a, b));

Point p1l = new Point(xo, yo + cl);

Point p2l = new Point(xo - cx, yo);

Point p1r = new Point(xo + cx, yo);

Point p2r = new Point(xo, yo - cl);

ArrayList<Point> res = new ArrayList<>();

res.add(new Point(xo, yo));

res.add(p1l);

res.add(p2l);

res.add(p1r);

res.add(p2r);

return res;

}

}

**Класс point :**

public class Point {

double x;

double y;

**// Конструктор точки**

Point(double x, double y) {

this.x = x;

this.y = y;

}

**//Получение случайной точки**

static Point getRandomPoint() {

Random r = new Random();

double nx = r.nextDouble() \* 2 - 1;

double ny = r.nextDouble() \* 2 - 1;

return new Point(nx, ny);

}

**// Рисование точки**

void render(GL2 gl) {

gl.glPointSize(3);

gl.glBegin(GL.GL\_POINTS);

gl.glVertex2d(x, y);

gl.glEnd();

gl.glPointSize(1);

}

**// Получение строкового представлениея точки**

@Override

public String toString() {

return "Точка с координатами: {" + x + "," + y + "}";

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Point point = (Point) o;

return isSolution == point.isSolution && Double.compare(point.x, x) == 0 && Double.compare(point.y, y) == 0;

}

@Override

public int hashCode() {

return Objects.hash(isSolution, x, y);

}

}

1. **Пример работы программы**

|  |  |
| --- | --- |
| Исходные данные | Выходные данные |
| -0,87 0,11 0,32 -0,29 0,44 -0,58 -0,45 -0,38 -0,01 -0,50 0,91 -0,20 0,37 0,51 -0,86 0,82 0,27 -0,44 -0,24 -0,93 -0,50 0,02 -0,60 0,79 0,30 0,30 -0,09 0,54 0,31 0,88 -0,55 0,41 -0,46 -0,56 0,69 -0,06 0,03 -0,80 0,22 0,48 0,76 0,73 -0,54 -0,10 0,83 0,13 -0,86 0,29 -0,68 0,05 0,59 0,31 0,53 0,32 -0,23 0,75 -0,16 0,54 0,94 -0,88 -0,61 0,05 0,10 -0,13 -0,95 0,74 0,51 0,34 -0,46 -0,47 0,52 0,62 -0,35 -0,23 -0,54 0,95 -0,90 -0,90 -0,48 0,43 0,26 0,71 0,10 0,77 0,00 0,72 -0,36 0,74 0,29 0,97 0,96 0,31 -0,47 0,32 -0,30 0,59 -0,46 -0,94 -0,15 -0,24 -0,85 -0,07 | 0,46 -0,63 -1,00 0,24 -0,52 0,28 0,06 0,68 0,41 0,61 0,52 1,00 0,02 -0,97 0,26 -0,24 0,40 0,75 0,69 0,94 0,71 0,98 0,80 -0,15 0,86 -0,59 -0,31 -0,65 -0,22 0,14 0,60 -0,33 -0,38 0,65 0,57 -0,51 0,14 0,66 0,49 -0,87 -0,81 -0,22 |

1. **Анализ правильности решения**

Можно с уверенностью сказать, что данная программа работает верно и решает поставленную задачу, так как при запуске программы с маленьким количеством точек в изначальном множестве она выдает верный результат, который совпадает с результатами, полученными ручным решением задачи.

Все точки, которые необходимо выделить, выделяются.

Программа верно строит прямую, проходящую через две выбранные точки.

Программа верно находит расстояния от каждой из точек до прямой и верно устанавливает из этих значений дистанцию.

Программа верно строит коридор – две построенные прямые действительно находятся от данной прямой на расстоянии, равном дистанции.