**Лабораторная работа № 1 “Пересечение двух отрезков”.**

**ЦЕЛЬ**

Разобраться в алгоритме определения пересечения двух отрезков на основе векторного произведения и запрограммировать его.

**РЕАЛИЗАЦИЯ**

Был реализован алгоритм (на основе векторного произведения) на языке Java в соответствии с псевдокодом из презентации (методы segments\_intersect, direction, on\_segment), позволяющий определить, пересекаются ли два отрезка (координаты точек начала и конца каждого отрезка вводятся пользователем).

**ИСПОЛЬЗОВАННЫЕ СТРУКТУРЫ ДАННЫХ**

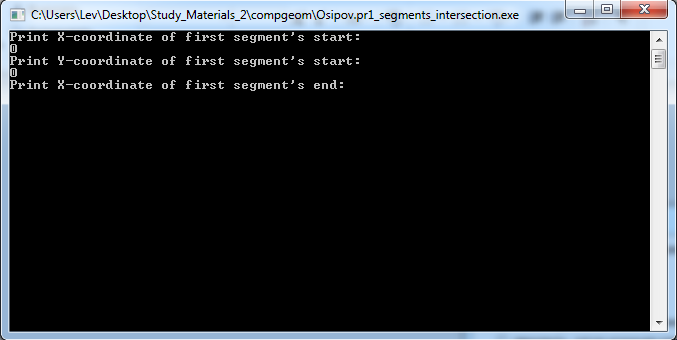
* Класс **Task1** – класс, в котором описаны основные методы алгоритма (segments\_intersect, direction, on\_segment), а также метод main, где происходит ввод, вывод и обрабатывание данных и выполнения алгоритма
* Класс **Point** – класс, моделирующий точку на плоскости. Содержит координаты X и Y, а также позволяет «вычитать» одну точку из другой (вычитая их координаты) – это нужно для нахождения координат вектора.
* Класс **Vector** – класс, моделирующий вектор. Содержит конструктор, который принимает точки начала и конца вектора и вычитает их, получая координаты вектора. Также в классе описана функция векторного произведения.

**ОЦЕНКА АЛГОРИТМА**

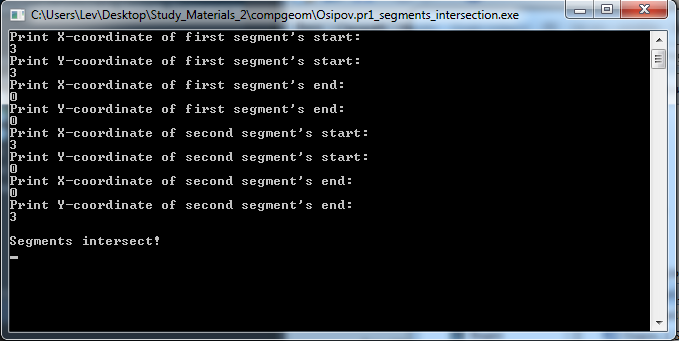
Асимптотической оценкой времени выполнения алгоритма является О(1), так как алгоритм не зависит от входных данных.

**СКРИНШОТЫ**

Скриншот ввода данных:



Скриншот вывода результата:



**ВЫВОДЫ**

Был разобран алгоритм о пересечении двух отрезков, который является примером использования векторного произведения. Задача о пересечении двух отрезков применяется в различных практических областях, так как является базовой и необходимой для решений более сложных задач в различных системах. В качестве примеров можно привести такие системы, как CAD и CAM.

**СПИСОК ИСТОЧНИКОВ**

Презентация «Вычислительная геометрия. Лекция 1. Введение», ВШЭ.

**ПРИЛОЖЕНИЕ (код программы)**

**Код файла Task1.java**

/\*\*

\* Created with IntelliJ IDEA.

\* User: Lev

\* Date: 22.01.14

\* Time: 10:56

\* To change this template use File | Settings | File Templates.

\*/

import java.util.Scanner;

public class Task1 {

public static void main(String[] args) {

double x1, x2, x3, x4, y1, y2, y3, y4;

Scanner input = new Scanner(System.in);

while (true) {

try {

System.out.println("Print X-coordinate of first segment's start:");

x1 = Double.parseDouble(input.next());

System.out.println("Print Y-coordinate of first segment's start:");

y1 = Double.parseDouble(input.next());

System.out.println("Print X-coordinate of first segment's end:");

x2 = Double.parseDouble(input.next());

System.out.println("Print Y-coordinate of first segment's end:");

y2 = Double.parseDouble(input.next());

System.out.println("Print X-coordinate of second segment's start:");

x3 = Double.parseDouble(input.next());

System.out.println("Print Y-coordinate of second segment's start:");

y3 = Double.parseDouble(input.next());

System.out.println("Print X-coordinate of second segment's end:");

x4 = Double.parseDouble(input.next());

System.out.println("Print Y-coordinate of second segment's end:");

y4 = Double.parseDouble(input.next());

break;

}

catch (NumberFormatException e) {

System.out.println("Input error!");

}

}

Point p1 = new Point(x1, y1);

Point p2 = new Point(x2, y2);

Point p3 = new Point(x3, y3);

Point p4 = new Point(x4, y4);

System.out.println();

String solution = segments\_intersect(p1, p2, p3, p4)? "Segments intersect!" : "Segments do not intersect!";

System.out.println(solution);

}

/\*\*

\* Определяет, как лежат относительно друг друга вектора на основе векторного произведения

\* @param pi точка начала первого и второго векторов

\* @param pj точка конца второго вектора

\* @param pk точка конца первого вектора

\* @return векторное произведение двух векторов

\*/

static double direction(Point pi, Point pj, Point pk) {

return (new Vector(pi, pk).composite(new Vector(pi, pj)));

}

/\*\*

\* Определяет, лежит ли точка на отрезке (при условии, что она коллинеарна ему)

\* @param pi первая точка отрезка

\* @param pj вторая точка отрезка

\* @param pk точка

\* @return лежит ли точка на отрезке (true или false)

\*/

static boolean on\_segment(Point pi, Point pj, Point pk) {

if ( Math.min(pi.getX(), pj.getX()) <= pk.getX() && Math.max(pi.getX(), pj.getX()) >= pk.getX() &&

Math.min(pi.getY(), pj.getY()) <= pk.getY() && Math.max(pi.getY(), pj.getY()) >= pk.getY() ) {

return true;

}

return false;

}

/\*\*

\* Определяет, пересекаются ли отрезки.

\* @param p1 первая точка первого отрезка

\* @param p2 вторая точка первого отрезка

\* @param p3 первая точка второго отрезка

\* @param p4 вторая точка второго отрезка

\* @return пересекаются ли отрезки (true или false)

\*/

static boolean segments\_intersect(Point p1, Point p2, Point p3, Point p4) {

double d1 = direction(p3, p4, p1);

double d2 = direction(p3, p4, p2);

double d3 = direction(p1, p2, p3);

double d4 = direction(p1, p2, p4);

if ( ( (d1 > 0 && d2 < 0) || (d1 < 0 && d2 > 0) ) && ( (d3 > 0 && d4 < 0) || (d3 < 0 && d4 > 0) ) ) {

return true;

}

if (d1 == 0 && on\_segment(p3, p4, p1)) {

return true;

}

if (d2 == 0 && on\_segment(p3, p4, p2)) {

return true;

}

if (d3 == 0 && on\_segment(p1, p2, p3)) {

return true;

}

if (d4 == 0 && on\_segment(p1, p2, p4)) {

return true;

}

return false;

}

}

/\*\*

\* Моделирует точку на плоскости.

\*/

class Point {

private double x;

private double y;

public Point(double x, double y) {

this.x = x;

this.y = y;

}

public double getX() {

return x;

}

public double getY() {

return y;

}

/\*\*

\* Вычитает координаты точек

\* @param op вычитаемое

\* @return this - op

\*/

public Point minus(Point op) {

return new Point(op.x - x, op.y - y);

}

}

/\*\*

\* Моделирует вектор.

\*/

class Vector {

/\*\*

\* Координаты вектора

\*/

private Point point;

/\*\*

\* Создает вектор из двух точек.

\* @param start точка начала

\* @param end точка конца

\*/

public Vector(Point start, Point end) {

point = start.minus(end);

}

/\*\*

\* Векторное произведение данного вектора на другой.

\* @param op вектор на который умножаем

\* @return векторное произведение

\*/

public double composite(Vector op) {

return point.getX() \* op.point.getY() - point.getY() \* op.point.getX();

}

}