**Лабораторная работа № 3 “Построение выпуклой оболочки по алгоритму Джарвиса”.**

**ЦЕЛЬ**

Разобраться в алгоритме построения выпуклой оболочки по алгоритму Джарвиса и запрограммировать его.

**РЕАЛИЗАЦИЯ**

Был реализован алгоритм на языке Java в общем соответствии с псевдокодом из Википедии:

**Jarvis**(P)

1) p[1] = самая левая нижняя точка множества P;

2) i = 1;

3) **do**:

p[i+1] = любая точка из P (кроме уже попавших в выпуклую оболочку, но включая p[1]);

(a)**for** (для каждой точки j из P, кроме уже попавших в выпуклую оболочку, но включая p[1])

p[i+1] = min\_polar\_angle(p[i+1], P[j]); // минимум через векторное произведение, как описано выше

(b)i = i + 1;

**while** p[i] != p[1]

4) **return** p

Пользовательский интерфейс построен так: пользователь вводит количество точек, которые генерируются случайным образом, и задержку в миллисекундах. Далее реализована пошаговая реализация алгоритма с задержкой, установленной пользователем. В любой момент выполнение программы можно остановить. Также реализовано добавление точек пользователем (в области рисования) по нажатию мыши и возможность очистки панели от точек.

**ИСПОЛЬЗОВАННЫЕ СТРУКТУРЫ ДАННЫХ**

* Класс **Task3** – класс с методом main, который является точкой входа в программу. Реализует элементы пользовательского интерфейса, обработку данных и запуск алгоритма и его визуализации.
* 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 и функция нахождения длины вектора.
* Класс **JarvisAlgorithm** – класс, содержащий в себе основные методы для работы алгоритма (случайную генерацию точек, нахождение самой нижней (и левой, если их несколько) точки, поиск следующей точки(с минимальным полярным углом относительно предыдущей), а также непосредственного сам алгоритм Джарвиса).
* Класс **DrawPanel** – класс, моделирующий визуализацию алгоритма (рисующий на панели точки и линии для пошагового отображения).

**ОЦЕНКА АЛГОРИТМА**

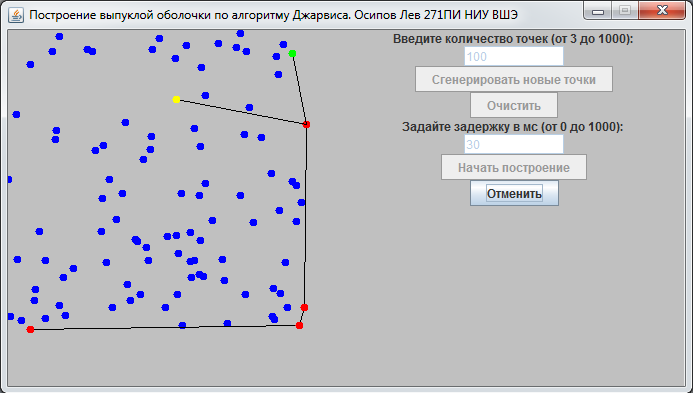
Асимптотической оценкой времени выполнения алгоритма является О(n\*h), где h – количество точек выпуклой оболочки. Это обусловлено тем, что h раз идет поиск следующей точки среди n точек.

**СКРИНШОТЫ**

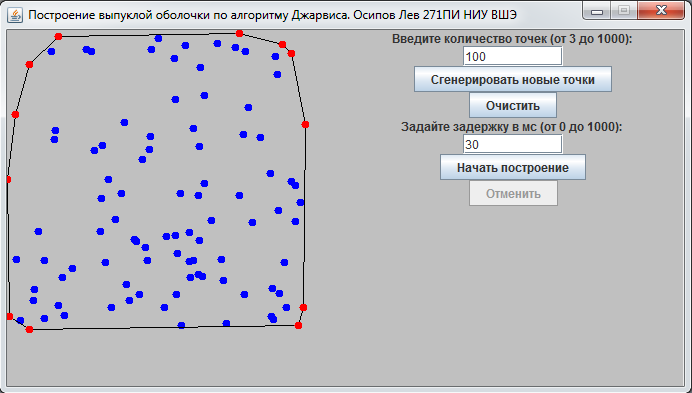
Скриншот ввода данных:



Скриншот процесса выполнения алгоритма:



Скриншот окончания работы алгоритма:



**ВЫВОДЫ**

Был разобран алгоритм Джарвиса построения выпуклой оболочки. В очередной раз была обнаружена полезность использования векторного произведения.

В отличие от алгоритма Грэхема, который является относительно стабильным по времени (n\*logn), время выполнения алгоритма Джарвиса больше зависит от входных данных, а потому может являться более или менее эффективным в различных случаях, то есть при выборе алгоритма для построения выпуклой оболочки нужно опираться на условие задачи.

Нахождение выпуклой оболочки является актуальной проблемой, так как используется в различных прикладных задачах.

**СПИСОК ИСТОЧНИКОВ**

1. Презентация «Вычислительная геометрия. Лекция 2. Построение выпуклой оболочки», ВШЭ.
2. Википедия: Алгоритм Джарвиса (http://ru.wikipedia.org/wiki/Алгоритм\_Джарвиса)

**ПРИЛОЖЕНИЕ (код программы)**

**Код файла Task3.java**

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.MouseEvent;

import java.awt.event.MouseListener;

import java.util.ArrayList;

import java.util.Stack;

/\*\*

\* Форма и кнопки

\*/

public class Task3 extends JFrame {

/\*\* Список точек \*/

private ArrayList<Point> list;

/\*\* Панель отрисовки \*/

final DrawPanel p = new DrawPanel();

/\*\* Поток выполнения и отрисовки алгоритма \*/

Thread graham;

final JTextField textCount = new JTextField();

final JButton buttonGen = new JButton("Сгенерировать новые точки");

final JTextField textDelay = new JTextField();

final JButton buttonStart = new JButton("Начать построение");

final JButton buttonClear = new JButton("Очистить");

final JButton buttonEnd = new JButton("Отменить");

public Task3() {

Toolkit tk = Toolkit.getDefaultToolkit();

Dimension d = tk.getScreenSize();

int w = d.width;

int h = d.height;

setSize(w / 2, h / 2);

setLocation(w / 4, h / 4);

setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

createGUI();

}

void createGUI() {

setLayout(new GridLayout(0, 2));

p.setPreferredSize(new Dimension(getWidth() / 2, getHeight()));

add(p);

final JPanel panel = new JPanel();

panel.setBackground(Color.LIGHT\_GRAY);

panel.setPreferredSize(new Dimension(this.getWidth() / 2, this.getHeight()));

this.add(panel);

panel.setLayout(new BoxLayout(panel, BoxLayout.Y\_AXIS));

JLabel labelCount = new JLabel("Введите количество точек (от 3 до 1000): ");

labelCount.setAlignmentX(Component.CENTER\_ALIGNMENT);

panel.add(labelCount);

textCount.setMaximumSize(new Dimension(100, 20));

textCount.setAlignmentX(Component.CENTER\_ALIGNMENT);

panel.add(textCount);

/\*\* Кнопка генерации новых точек \*/

buttonGen.setAlignmentX(Component.CENTER\_ALIGNMENT);

buttonGen.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

p.setPoints(null);

p.setEnd(false);

int count;

if (textCount.getBackground() != Color.WHITE) {

textCount.setBackground(Color.WHITE);

}

try {

count = Integer.parseInt(textCount.getText());

if (count < 3 || count > 1000) {

throw new NumberFormatException();

}

}

catch (NumberFormatException ex) {

textCount.setBackground(Color.PINK);

return;

}

list = JarvisAlgorithm.generatePoints(count);

p.setAllPoints(list);

p.repaint();

}

});

panel.add(buttonGen);

/\*\* Кнопка генерации новых точек \*/

buttonClear.setAlignmentX(Component.CENTER\_ALIGNMENT);

buttonClear.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

p.setPoints(null);

p.setEnd(false);

list = null;

p.setAllPoints(list);

p.repaint();

}

});

panel.add(buttonClear);

JLabel labelDelay = new JLabel("Задайте задержку в мс (от 0 до 1000): ");

labelDelay.setAlignmentX(Component.CENTER\_ALIGNMENT);

panel.add(labelDelay);

textDelay.setMaximumSize(new Dimension(100, 20));

textCount.setAlignmentX(Component.CENTER\_ALIGNMENT);

panel.add(textDelay);

/\*\* Кнопка запуска алгоритма \*/

buttonStart.setAlignmentX(Component.CENTER\_ALIGNMENT);

buttonStart.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

if (list == null) {

JOptionPane.showMessageDialog(null, "Точек нет!");

return;

}

graham = new Thread() {

@Override

public void run() {

buttonStart.setEnabled(false);

textCount.setEnabled(false);

textDelay.setEnabled(false);

buttonGen.setEnabled(false);

buttonClear.setEnabled(false);

p.setEnabled(false);

buttonEnd.setEnabled(true);

p.setEnd(false);

while (buttonStart.isEnabled()) { }

int delay;

if (textDelay.getBackground() != Color.WHITE) {

textDelay.setBackground(Color.WHITE);

}

try {

delay = Integer.parseInt(textDelay.getText());

if (delay < 0 || delay > 1000) {

throw new NumberFormatException();

}

}

catch (NumberFormatException ex) {

textDelay.setBackground(Color.PINK);

return;

}

JarvisAlgorithm alg = new JarvisAlgorithm();

ArrayList<Point> tempList = new ArrayList<Point>(list);

Stack<Point> sol = alg.scanJarvis(tempList, p, delay);

textCount.setEnabled(true);

textDelay.setEnabled(true);

buttonGen.setEnabled(true);

p.setEnabled(true);

buttonStart.setEnabled(true);

buttonClear.setEnabled(true);

p.setEnabled(true);

buttonEnd.setEnabled(false);

}

};

graham.start();

}

});

panel.add(buttonStart);

/\*\* Кнопка отмены выполнения \*/

buttonEnd.setAlignmentX(Component.CENTER\_ALIGNMENT);

buttonEnd.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

graham.interrupt();

try {

graham.join();

}

catch(InterruptedException e1) {

e1.printStackTrace();

}

textCount.setEnabled(true);

textDelay.setEnabled(true);

buttonGen.setEnabled(true);

buttonClear.setEnabled(true);

buttonStart.setEnabled(true);

}

});

panel.add(buttonEnd);

/\*\* Добавление точек по клику \*/

p.addMouseListener(new MouseListener() {

@Override

public void mouseClicked(MouseEvent e) {

}

@Override

public void mousePressed(MouseEvent e) {

}

@Override

public void mouseReleased(MouseEvent e) {

/\*\* Если не идет процесс выполнения и в пределах поля отрисовки \*/

if (p.isEnabled() && e.getX() <= DrawPanel.getSIZE() && e.getY() <= DrawPanel.getSIZE() &&

e.getX() >= 0 && e.getY() >= 0) {

if (list == null) {

list = new ArrayList<Point>();

}

list.add(new Point(e.getX(), 300 - e.getY()));

p.setAllPoints(list);

p.repaint();

}

}

@Override

public void mouseEntered(MouseEvent e) {

}

@Override

public void mouseExited(MouseEvent e) {

}

});

/\*\* Изначальные действия \*/

buttonEnd.setEnabled(false);

textCount.setText("100");

textDelay.setText("30");

p.setPoints(null);

p.setEnd(false);

list = JarvisAlgorithm.generatePoints(Integer.parseInt(textCount.getText()));

p.setAllPoints(list);

p.validate();

p.repaint();

}

public static void main(String[] args) {

/\*\* Запуск \*/

Task3 f = new Task3();

f.setVisible(true);

f.setResizable(false);

f.setTitle("Построение выпуклой оболочки по алгоритму Джарвиса. Осипов Лев 271ПИ НИУ ВШЭ");

}

}**Код файла Point.java**

public 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;

}

public Point minus(Point op) {

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

}

@Override

public String toString() {

return "(" + x + " , " + y + ")";

}

}

**Код файла JarvisAlgoritm.java**

import java.util.\*;

/\*\*

\* Created by Lev on 02.02.14.

\*/

public class JarvisAlgorithm {

/\*\* Самая нижняя (и левая, если их несколько) \*/

private Point pBot;

/\*\*

\* Генерирует список точек по их заданному количеству

\* @param count количество

\* @return список точек

\*/

public static ArrayList<Point> generatePoints(int count) {

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

Random gen = new Random();

for (int i = 0; i < count; ++ i) {

list.add(new Point(gen.nextInt(DrawPanel.getSIZE()), gen.nextInt(DrawPanel.getSIZE())));

}

return list;

}

/\*\*

\* Ищет выпуклую оболочку сканированием Грэхема

\* @param Q список точек

\* @return CH(Q)

\*/

public Stack<Point> scanJarvis(List<Point> Q, DrawPanel panel, int delay) {

Stack<Point> ch = new Stack<Point>();

/\*\* Нахождение нижней \*/

pBot = findBotPoint(Q);

ch.push(pBot);

panel.setPoints(ch);

panel.repaint();

/\*\* Алгоритм \*/

Point current;

do {

try {

Thread.sleep(delay);

current = findNext(Q, ch, panel, delay);

}

catch(InterruptedException e) {return null;}

panel.repaint();

} while (!current.equal(pBot));

panel.setEnd(true);

panel.repaint();

return ch;

}

public Point findNext(List<Point> src, Stack<Point> ch, DrawPanel p, int delay)

throws InterruptedException {

int ind = 0;

Point current = ch.peek(); // очередная точка

double comp;

for (int i = 1; i < src.size(); ) {

p.setTempMinPoint(src.get(ind));

p.setTempPoint(src.get(i));

p.repaint();

Thread.sleep(delay);

Vector v1 = new Vector(current, src.get(ind));

Vector v2 = new Vector(current, src.get(i));

comp = v1.composite(v2);

if (comp < 0) {

ind = i;

}

if (comp == 0 && !current.equal(src.get(i))) {

if (v1.length() > v2.length()) {

src.remove(i);

}

else {

src.remove(ind);

ind = i - 1;

}

continue;

}

++ i;

}

ch.push(src.get(ind));

return src.get(ind);

}

/\*\*

\* Поиск самой нижней точки (если таких несколько, то самой левой)

\* @param src список точек

\*/

public Point findBotPoint(List<Point> src) {

double minX = src.get(0).getX(), minY = src.get(0).getY();

int ind = 0;

for (int i = 0; i < src.size(); ++ i) {

if (src.get(i).getY() < minY || (src.get(i).getY() == minY && src.get(i).getX() < minX)) { // ниже, либо так же, но левее

minX = src.get(i).getX();

minY = src.get(i).getY();

ind = i;

}

}

return src.get(ind);

}

}

/\*\*

\* Моделирует вектор.

\*/

class Vector {

private Point point; // координаты вектора

public Vector(Point start, Point end) {

point = start.minus(end);

}

public double composite(Vector op) {

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

}

public double length() {

return Math.sqrt(point.getX() \* point.getX() + point.getY() \* point.getY());

}

}

**Код файла DrawPanel.java**

import javax.swing.\*;

import java.awt.\*;

import java.util.ArrayList;

import java.util.Stack;

/\*\*

\* Рисование области точек и выпуклой оболочки.

\*/

public class DrawPanel extends JPanel {

/\*\* Размер области для рисования \*/

private static final int SIZE = 300;

DrawPanel() {

setBackground(Color.LIGHT\_GRAY);

}

/\*\* Все точки \*/

private ArrayList<Point> all;

/\*\* Точки оболочки \*/

private Stack<Point> st;

/\*\* Временная точка \*/

private Point tempPoint;

/\*\* Временная минимальная точка (с минимальным полярным углом)\*/

private Point tempMinPoint;

/\*\* Рисовать ли последнюю линию \*/

private boolean end = false;

public void setPoints(Stack<Point> st) {

this.st = st;

}

public void setAllPoints(ArrayList<Point> all) {

this.all = all;

}

public void setEnd(boolean end) {

this.end = end;

}

public void setTempPoint(Point tempPoint) {

this.tempPoint = tempPoint;

}

public void setTempMinPoint(Point tempMinPoint) {

this.tempMinPoint = tempMinPoint;

}

public static int getSIZE() {

return SIZE;

}

/\*\*

\* Рисует точки и линии в зависимости от состояния алгоритма.

\* Координаты центра точек смещены.

\* СК перевернута.

\*/

@Override

public void paintComponent(Graphics g) {

super.paintComponent(g);

/\*\* Отрисовка линий - границ \*/

/\* g.setColor(Color.DARK\_GRAY);

g.drawLine(0, SIZE, SIZE, SIZE);

g.drawLine(SIZE, 0, SIZE, SIZE); \*/

/\*\* Отрисовка всех точек множества \*/

g.setColor(Color.BLUE);

if (all != null) {

for (Point p : all) {

g.fillOval((int)p.getX() - 4, SIZE - (int)p.getY() - 4, 8, 8);

}

}

/\*\* Отрисовка линий между точками ch \*/

g.setColor(Color.BLACK);

if (st != null) {

for (int i = 1; i < st.size(); ++ i) {

g.drawLine((int)st.get(i).getX(), SIZE - (int)st.get(i).getY(),

(int)st.get(i - 1).getX(),SIZE - (int)st.get(i - 1).getY());

}

/\*\* Отрисовка последней линии \*/

if (end) {

g.drawLine((int)st.peek().getX(), SIZE - (int)st.peek().getY(),

(int)st.get(0).getX(), SIZE - (int)st.get(0).getY());

tempPoint = null;

tempMinPoint = null;

}

/\*\* Закрашивание точек ch \*/

g.setColor(Color.RED);

for (int i = 0; i < st.size(); ++ i) { // красные точки

g.fillOval((int) st.get(i).getX() - 4, SIZE - (int) st.get(i).getY() - 4, 8, 8);

}

/\*\* Отрисовка временной точки и линии к ней \*/

if (tempPoint != null) {

g.setColor(Color.BLACK);

g.drawLine((int)st.peek().getX(), SIZE - (int)st.peek().getY(),

(int)tempPoint.getX(),SIZE - (int)tempPoint.getY());

g.setColor(Color.YELLOW);

g.fillOval((int) tempPoint.getX() - 4, SIZE - (int) tempPoint.getY() - 4, 8, 8);

}

/\*\* Отрисовка временной минимальной точки и линии к ней \*/

if (tempMinPoint != null) {

g.setColor(Color.BLACK);

g.drawLine((int)st.peek().getX(), SIZE - (int)st.peek().getY(),

(int)tempMinPoint.getX(),SIZE - (int)tempMinPoint.getY());

g.setColor(Color.GREEN);

g.fillOval((int) tempMinPoint.getX() - 4, SIZE - (int) tempMinPoint.getY() - 4, 8, 8);

}

}

}

}

}

}

}