МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«Санкт–Петербургский государственный университет
аэрокосмического приборостроения»

Кафедра №43 «Компьютерных технологий и программной инженерии»

ОТЧЁТ ПО ПРАКТИКЕ
ЗАЩИЩЁН С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

Ст. преподаватель		С.А. Рогачев
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЁТ ПО ПРАКТИКЕ

вид практики производственная

тип практики технологическая (проектно-технологическая)

на тему индивидуального задания Программная реализация алгоритма построения диаграммы Вороного, для случайных точек плоскости

выполнен Кресик Елизаветой Александровной
фамилия, имя, отчество обучающегося в творительном падеже

по направлению подготовки 09.03.04 (код) Программная инженерия (наименование направления)

наименование направления

направленности 2 (код) наименование направленности

Проектирование программных систем
наименование направленности

Обучающийся группы № 4131 (номер) 18.07.2023 (подпись, дата) Е.А.Кресик (инициалы, фамилия)

Санкт–Петербург 2023

СОДЕРЖАНИЕ

1 Цель работы

Целью работы является программная реализация алгоритма построения диаграммы Вороного, для случайных точек плоскости.

2 Исходные данные

1. Диаграмма Вороного - https://ru.wikipedia.org/wiki/Диаграмма_Вороного
2. Диаграмма Вороного и её применения - https://habr.com/ru/articles/309252/

3 Теория

Диаграмма Вороного конечного множества точек S на плоскости представляет такое разбиение плоскости, при котором каждая область (локус) этого разбиения образует множество точек, более близких к одному из элементов множества S, чем к любому другому элементу множества.

Алгоритмы построения:

1. Простой алгоритм. Вычислительная сложность $O(n^4)$
2. Рекурсивный алгоритм. Вычислительная сложность O(n*log(n))
3. Алгоритм Форчуна. Вычислительная сложность O(n*log(n))

Для своей программы я выбрала Простой алгоритм.

В данном алгоритме последовательно для каждого сайта (так называется точка, для которой строится локус) будет находиться множество точек, при последовательном соединении которых получится искомый локус.

Процесс нахождения вершин локуса для каждого сайта состоит из 3-х пунктов:

1. Нахождение всех серединных перпендикуляров между рассматриваемым сайтом и всеми остальными точками из входных данных. Серединный перпендикуляр – прямая, перпендикулярная отрезку, соединяющему сайт и рассматриваемую точку, проходящая через середину этого отрезка.
2. Нахождение точек пересечения серединных перпендикуляров между собой, а также с прямыми, ограничивающими плоскость (область видимости).
3. Отбор точек, лежащих в одной полуплоскости с рассматриваемым сайтом относительно каждого серединного перпендикуляра. Найденные точки и будут являться вершинами локуса.

Данный процесс необходимо проделать для каждой точки из входных данных.

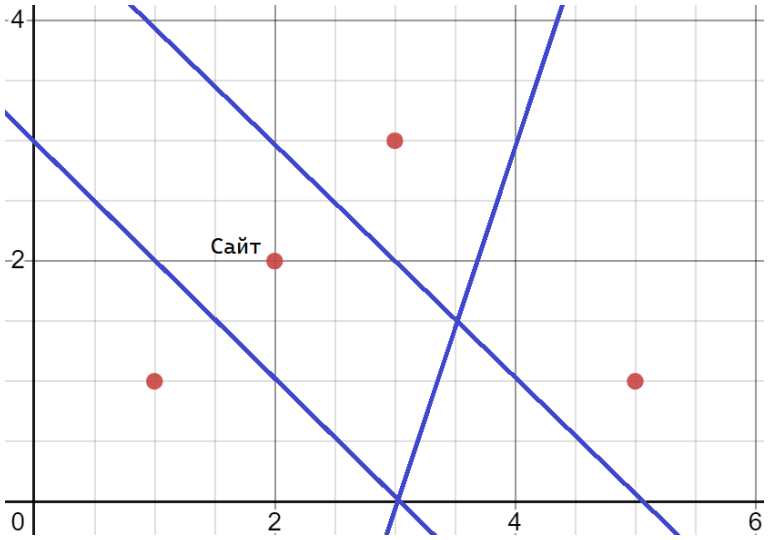

Рисунок 1. Иллюстрация Простого метода, этап 1

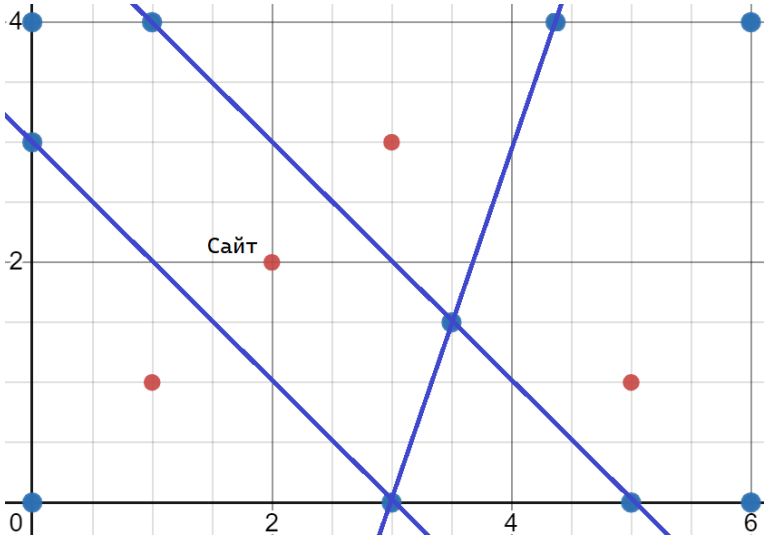

Рисунок 2. Иллюстрация Простого метода, этап 2

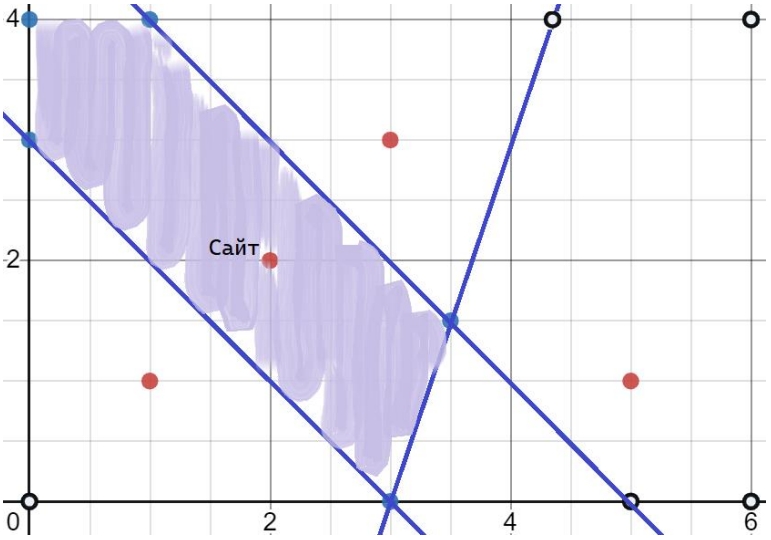

Рисунок 3. Иллюстрация Простого метода, этап 3

4 Практика

Язык разработки был выбран C++, для достижения максимального быстродействия программы. Также для вывода диаграммы была использована библиотека SFML.

4.1 Структуры

Структура **Point** – хранение координат точки (x и y).

Структура **Line** – хранения коэффициентов уравнения прямой

Уравнение прямой y = k*x+b. В структуре хранятся коэффициенты k и b. Также есть поля x и y – они равны 0, но заполняются значениями при условии, что прямая параллельна одной из осей координат, при этом поля k и b равны 0.

4.2 Функции

medianPerpendicularsForSite – функция получения серединных перпендикуляров.

```
void medianPerpendicularsForSite(Point point, vector <Point> points, vector <Line>* medianPerpendiculars);
```

intersectionsPerpendicularsForSite – функция получения точек пересечения серединных перпендикуляров.

```
void intersectionsPerpendicularsForSite(const vector <Line>* const medianPerpendiculars, vector <Point>* intersectionsPerpendiculars);
```

pointsForLocusOfSite – функция отбора точек, лежащих в одной полуплоскости с рассматриваемым сайтом.

```
void pointsForLocusOfSite(const vector <Line>* const medianPerpendiculars, const vector <Point>* const intersectionsPerpendiculars, vector <Point>* const pointsForLocus, Point point);
```

sort – функция сортировки вершин локуса в последовательном порядке.

```
void sort(vector <Point>* const pointsForLocus, Point point);
```

determineLocuses – функция, возвращающая вектор векторов, хранящих вершины каждого локуса.

```
vector <vector <Point>> determineLocuses(vector <Point> points)
```

removeRepetitions – функция удаления повторяющихся точек из вектора.

```
void removeRepetitions(vector <Point>* vec)
```

5 Результаты

Программа позволяет генерировать количество точек и сами точки. Программа выводит диаграмму в дополнительном окне.

Рисунок 4 – Пример вывода программы

```
Выводим нагенерированные точки
2 2
2 9
4 2
8 3
5 3
1 9
8 1
2 8
7 6
8 9
5 5
6 5
1 1
4 9
3 5
8 7
```

Рисунок 5 – Вывод текстовой информации

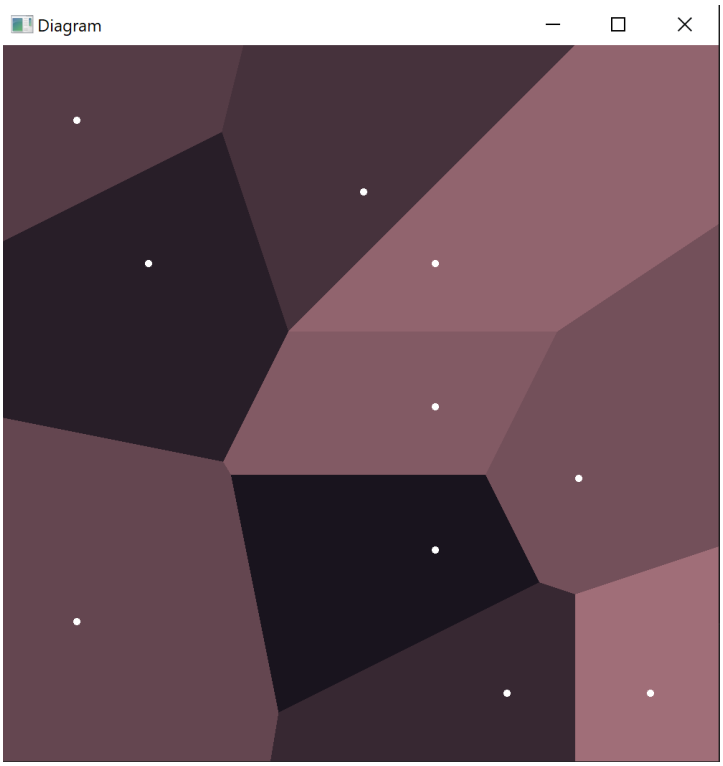

Рисунок 6 – Пример вывода программы

```
Выводим нагенерированные точки
6 7
2 3
7 9
5 2
1 1
1 8
8 6
6 5
6 3
9 9
```

Рисунок 7 – Вывод текстовой информации

ЗАКЛЮЧЕНИЕ

Результатом практики является программа, конструирующая диаграмму Вороного для случайных точек на плоскости. Также, в ходе практики я улучшила свои навыки программирования на c++, познакомилась с библиотекой SFML.

СПИСОК ЛИТЕРАТУРЫ

1. Препарата Ф., Шеймос М. Вычислительная геометрия. Введение (1989);
2. https://www.sfml-dev.org/tutorials/2.6/
3. https://habr.com/ru/articles/309252/
4. C++ за 21 день. Сиддхартх Рао.
5. Язык программирования C++. Базовый курс. Стенли Б. Липпман.

ПРИЛОЖЕНИЕ

```
#include <SFML/Graphics.hpp>
#include <iostream>
#include <vector>
#include <cmath>
#define PI 3.14159265


using namespace std;

struct Point
{
    double x;
    double y;

    Point() {};
    ~Point() {};
    Point(double xx, double yy)
    {
        x = xx;
        y = yy;
    }
};

struct Line //y = kx + b - уравнение прямой
{
    double k;
    double b;

    //для случаев, когда прямая параллельна одной из осей
    double x;
    double y;

    Line(double kk, double bb)
    {
        k = kk;
        b = bb;
        x = 0;
        y = 0;
    }
    Line() //конструктор по умолчанию
    {
        k = 0;
        b = 0;
        x = 0;
        y = 0;
    }
    ~Line() {};
};

bool operator == (const Line& a, const Line& b)
{
    if (a.x == b.x && a.y == b.y && a.k == b.k && a.b == b.b)
        return true;
    return false;
}

bool operator < (const Point& a, const Point& b)
{
    if (a.y != b.y)
        return a.y < b.y;
    return a.x < b.x;
```

```
}

void medianPerpendicularsForSite(Point point, vector <Point> points, vector
<Line>* medianPerpendiculars)
{
    //Получение (n-1) серединных перпендикуляров для данного сайта
    for (int j = 0; j < points.size(); j++)
    {
        if (points[j].x != point.x || points[j].y != point.y)
        {
            Point M; //Середина отрезка, соединяющего рассматриваемые точки
            M.x = (point.x + points[j].x) / 2;
            M.y = (point.y + points[j].y) / 2;

            Line AB; //Прямая, соединяющая точки
            if (point.x != points[j].x && point.y != points[j].y)
            {
                AB.k = (point.y - points[j].y) / (point.x - points[j].x);
                //AB.b = (points[i].y * points[j].x - points[j].y *
points[i].x) / (points[j].x - points[i].x);

                Line perpendicular;
                perpendicular.k = -1 / AB.k;
                perpendicular.b = M.y + M.x / AB.k;

                (*medianPerpendiculars).push_back(perpendicular);
            }
            else
            {
                if (point.x == points[j].x)
                {
                    AB.x = point.x; //Значит прямая, соединяющая сайт и
рассматриваемую точку - параллельна OY, x = AB.x
                    //А серпер будет || OX
                    Line perpendicular;
                    perpendicular.k = 0;
                    perpendicular.b = 0;
                    perpendicular.y = M.y;

                    (*medianPerpendiculars).push_back(perpendicular);
                }
                else
                {
                    AB.y = point.y; //Значит прямая, соединяющая сайт и
рассматриваемую точку - параллельна OX, y = AB.y
                    //А серпер будет || OY
                    Line perpendicular;
                    perpendicular.k = 0;
                    perpendicular.b = 0;
                    perpendicular.x = M.x;

                    (*medianPerpendiculars).push_back(perpendicular);
                }
            }
        }
    }
    Line line;
    line.x = 10;
    line.y = 10;
    (*medianPerpendiculars).push_back(line); //Добавляем по сути сразу две
линии - ограничения нашего поля по X и по Y

    (*medianPerpendiculars).push_back(Line()); //Добавляем по сути сразу две
линии - ОСИ координат
```

```
}

void removeRepetitions(vector <Point>* vec)
{
    //Удаляем повторившиеся точки (при попадании серпера в угол происходит дублирование точек)

    for (int i = 0; i < (*vec).size() - 1; i++)
    {
        for (int j = i + 1; j < (*vec).size(); j++)
        {
            if ((*vec)[i].x == (*vec)[j].x &&
                (*vec)[i].y == (*vec)[j].y && i != j)
            {
                auto it = (*vec).begin() + j;
                (*vec).erase(it);
                j -= 1;
            }
        }
    }
}

void intersectionsPerpendicularsForSite(const vector <Line>* const medianPerpendiculars, vector <Point>* intersectionsPerpendiculars)
{
    //Line two(-3, 15);

    //Получение точек пересечения серединных перпендикуляров между собой и с осями координат
    for (int k = 0; k < (*medianPerpendiculars).size() - 1; k++)
    {
        for (int f = k + 1; f < (*medianPerpendiculars).size(); f++)
        {
            if (((*medianPerpendiculars)[k].k != (*medianPerpendiculars)[f].k) ||
            (
                ((*medianPerpendiculars)[k].k == 0) && ((*medianPerpendiculars)[f].k == 0) &&
                (
                    (
                    (*medianPerpendiculars)[k].x == 0 && (*medianPerpendiculars)[f].x != 0 ||
                    (*medianPerpendiculars)[k].y == 0 && (*medianPerpendiculars)[f].y != 0 ||
                    (*medianPerpendiculars)[f].x == 0 && (*medianPerpendiculars)[k].x != 0 ||
                    (*medianPerpendiculars)[f].y == 0 && (*medianPerpendiculars)[k].y != 0
                    )
                )
            )
                )
                //Значит прямые пересекаются (условие параллельности прямых)
            {
                if ((*medianPerpendiculars)[f].x == 0 && (*medianPerpendiculars)[f].y == 0 && (*medianPerpendiculars)[f].k == 0 && (*medianPerpendiculars)[f].b == 0)
                {
                    if ((*medianPerpendiculars)[k].x != 0) //Значит мы сейчас проверяем пересечение прямой, параллельной OY с осями
                    {
                        Point intersection; //Точка (x, 0)
                        intersection.x = (*medianPerpendiculars)[k].x;
                        intersection.y = 0;
```

```
                        if (intersection.x >= 0 && intersection.y >= 0)
(*intersectionsPerpendiculars).push_back(intersection);

                        //Для варианта, в котором и x и y - не 0
                        if ((*medianPerpendiculars)[k].y != 0) //Значит мы
сейчас проверяем пересечение прямой, параллельной OX с осями
                        {
                            //Точка (0, y)
                            intersection.x = 0;
                            intersection.y = (*medianPerpendiculars)[k].y;
                            if (intersection.x >= 0 && intersection.y >= 0)
(*intersectionsPerpendiculars).push_back(intersection);
                        }
                    }
                    else
                    {
                        if ((*medianPerpendiculars)[k].y != 0) //Значит мы
сейчас проверяем пересечение прямой, параллельной OX с осями
                        {
                            Point intersection; //Точка (0, y)
                            intersection.x = 0;
                            intersection.y = (*medianPerpendiculars)[k].y;
                            if (intersection.x >= 0 && intersection.y >= 0)
(*intersectionsPerpendiculars).push_back(intersection);
                        }
                        else //Значит мы сейчас проверяем пересечение обычной
прямой, не параллельной осям, с осями
                        {
                            if ((*medianPerpendiculars)[k].b == 0)
(*intersectionsPerpendiculars).push_back(Point(0, 0)); //Точка (0, 0)
                            else
                            {
                                Point intersection; //Точка (0, y)
                                intersection.x = 0;
                                intersection.y =
(*medianPerpendiculars)[k].b;
                                if (intersection.x >= 0 && intersection.y >=
0)
(*intersectionsPerpendiculars).push_back(intersection);

                                //Точка (x, 0)
                                intersection.x = -
(*medianPerpendiculars)[k].b / (*medianPerpendiculars)[k].k;
                                intersection.y = 0;
                                if (intersection.x >= 0 && intersection.y >=
0)
(*intersectionsPerpendiculars).push_back(intersection);
                            }
                        }
                    }
                }
                else //Значит обе прямые - не оси
                {
                    if ((*medianPerpendiculars)[k].x != 0) //Значит мы сейчас
проверяем пересечение прямой, параллельной OY
                    {
                        if ((*medianPerpendiculars)[k].y != 0) //Для варианта
в котором x и y не 0
```

```
                        {
                            //С обычной прямой, не параллельной осям
                            Point intersection; //Точка (x, ?)
                            intersection.x = (*medianPerpendiculars)[k].x;
                            intersection.y = (*medianPerpendiculars)[f].k *
(*medianPerpendiculars)[k].x + (*medianPerpendiculars)[f].b;
                            if (intersection.x >= 0 && intersection.y >= 0)

(*intersectionsPerpendiculars).push_back(intersection);

                            //Точка (?, y)
                            intersection.x = ((*medianPerpendiculars)[k].y -
(*medianPerpendiculars)[f].b) / (*medianPerpendiculars)[f].k;
                            intersection.y = (*medianPerpendiculars)[k].y;
                            if (intersection.x >= 0 && intersection.y >= 0)

(*intersectionsPerpendiculars).push_back(intersection);
                        }
                        else
                        {
                            if ((*medianPerpendiculars)[f].y != 0) //С
прямой, параллельной OX
                            {
                                Point intersection; //Точка (x, y)
                                intersection.x =
(*medianPerpendiculars)[k].x;
                                intersection.y =
(*medianPerpendiculars)[f].y;
                                if (intersection.x >= 0 && intersection.y >=
0)

(*intersectionsPerpendiculars).push_back(intersection);
                            }
                            else //С обычной прямой, не параллельной осям
                            {
                                Point intersection; //Точка (x, ?)
                                intersection.x =
(*medianPerpendiculars)[k].x;
                                intersection.y = (*medianPerpendiculars)[f].k
* (*medianPerpendiculars)[k].x + (*medianPerpendiculars)[f].b;
                                if (intersection.x >= 0 && intersection.y >=
0)

(*intersectionsPerpendiculars).push_back(intersection);
                            }
                        }
                    }
                    else
                    {
                        if ((*medianPerpendiculars)[k].y != 0) //Значит мы
сейчас проверяем пересечение прямой, параллельной OX
                        {
                            if ((*medianPerpendiculars)[f].x != 0) //С
прямой, параллельной OY
                            {
                                Point intersection; //Точка (x, y)
                                intersection.x =
(*medianPerpendiculars)[f].x;
                                intersection.y =
(*medianPerpendiculars)[k].y;
                                if (intersection.x >= 0 && intersection.y >=
0)

(*intersectionsPerpendiculars).push_back(intersection);
```

```
                            }
                            else //С обычной прямой, не параллельной осям
                            {
                                Point intersection; //Точка (?, y)
                                intersection.x =
((*medianPerpendiculars)[k].y - (*medianPerpendiculars)[f].b) /
(*medianPerpendiculars)[f].k;
                                intersection.y =
(*medianPerpendiculars)[k].y;
                                if (intersection.x >= 0 && intersection.y >=
0)

(*intersectionsPerpendiculars).push_back(intersection);
                            }
                        }
                        else //Значит мы сейчас проверяем пересечение обычной
прямой, не параллельной осям
                        {
                            if ((*medianPerpendiculars)[f].x != 0) //С
прямой, параллельной OY
                            {
                                Point intersection; //Точка (x, ?)
                                intersection.x =
(*medianPerpendiculars)[f].x;
                                intersection.y = (*medianPerpendiculars)[k].k
* (*medianPerpendiculars)[f].x + (*medianPerpendiculars)[k].b;
                                if (intersection.x >= 0 && intersection.y >=
0)

(*intersectionsPerpendiculars).push_back(intersection);

                                //Для случаев, когда x != 0 и y != 0
                                if ((*medianPerpendiculars)[f].y != 0) //С
прямой, параллельной OX
                                {
                                    //Точка (?, y)
                                    intersection.x =
((*medianPerpendiculars)[f].y - (*medianPerpendiculars)[k].b) /
(*medianPerpendiculars)[k].k;
                                    intersection.y =
(*medianPerpendiculars)[f].y;
                                    if (intersection.x >= 0 && intersection.y
>= 0)

(*intersectionsPerpendiculars).push_back(intersection);
                                }
                            }
                            else
                            {
                                if ((*medianPerpendiculars)[f].y != 0) //С
прямой, параллельной OX
                                {
                                    Point intersection; //Точка (?, y)
                                    intersection.x =
((*medianPerpendiculars)[f].y - (*medianPerpendiculars)[k].b) /
(*medianPerpendiculars)[k].k;
                                    intersection.y =
(*medianPerpendiculars)[f].y;
                                    if (intersection.x >= 0 && intersection.y
>= 0)

(*intersectionsPerpendiculars).push_back(intersection);
                                }
                                else //С обычной прямой, не параллельной осям
```

```
                                    {
                                        //Точка (?, ?)
                                        Point intersection;
                                        intersection.x =
((*medianPerpendiculars)[k].b - (*medianPerpendiculars)[f].b) /
((*medianPerpendiculars)[f].k - (*medianPerpendiculars)[k].k);
                                        intersection.y = intersection.x *
(*medianPerpendiculars)[k].k + (*medianPerpendiculars)[k].b;
                                        if (intersection.x >= 0 && intersection.y
>= 0)

(*intersectionsPerpendiculars).push_back(intersection);
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    (*intersectionsPerpendiculars).push_back(Point(0, 0)); //Добавляем точку
середины координат
    (*intersectionsPerpendiculars).push_back(Point(10, 10)); //Добавляем
точку пересечения ограничивающих прямых

    //Удаляем повторившиеся точки (при попадании серпера в угол происходит
дублирование точек)
    removeRepetitions(intersectionsPerpendiculars);
    removeRepetitions(intersectionsPerpendiculars);
}



void pointsForLocusOfSite(const vector <Line>* const medianPerpendiculars,
const vector <Point>* const intersectionsPerpendiculars,
                          vector <Point>* const pointsForLocus, Point
point)
{
    //Проверка точек на пренадлежность всем полуплоскостям - для выявления
точек для локуса
    bool flag = false;

    for (int f = 0; f < (*intersectionsPerpendiculars).size(); f++)
//Проверяем каждую точку, по одну ли она сторону с текущим сайтом
    {
        for (int j = 0; j < (*medianPerpendiculars).size() - 1; j++) //Не
проверяем оси координат
        {
            if ((*medianPerpendiculars)[j].x != 0) //Значит прямая || OY
            {
                if ((point.x <= (*medianPerpendiculars)[j].x &&
(*intersectionsPerpendiculars)[f].x > (*medianPerpendiculars)[j].x) ||
                    (point.x >= (*medianPerpendiculars)[j].x &&
(*intersectionsPerpendiculars)[f].x < (*medianPerpendiculars)[j].x))
                {
                    flag = true;
                    break;
                }
                //Для варианта, в котором и x и y - не 0
                if ((*medianPerpendiculars)[j].y != 0) //Значит прямая || OX
                {
                    if ((point.y <= (*medianPerpendiculars)[j].y &&
(*intersectionsPerpendiculars)[f].y > (*medianPerpendiculars)[j].y) ||
```

```
                        (point.y >= (*medianPerpendiculars)[j].y &&
(*intersectionsPerpendiculars)[f].y < (*medianPerpendiculars)[j].y))
                    {
                        flag = true;
                        break;
                    }
                }
            }
            else
            {
                if ((*medianPerpendiculars)[j].y != 0) //Значит прямая || OX
                {
                    if ((point.y <= (*medianPerpendiculars)[j].y &&
(*intersectionsPerpendiculars)[f].y > (*medianPerpendiculars)[j].y) ||
                        (point.y >= (*medianPerpendiculars)[j].y &&
(*intersectionsPerpendiculars)[f].y < (*medianPerpendiculars)[j].y))
                    {
                        flag = true;
                        break;
                    }
                }
                else //Значит обычная прямая, не параллельная осям
                {
                    double a = (*medianPerpendiculars)[j].k * point.x -
point.y + (*medianPerpendiculars)[j].b;
                    double b = (*medianPerpendiculars)[j].k *
(*intersectionsPerpendiculars)[f].x - (*intersectionsPerpendiculars)[f].y +
(*medianPerpendiculars)[j].b;
                    if ((a >= 0 && b < -0.01) || (a <= 0 && b > 0.01))
                    {
                        flag = true;
                        break;
                    }
                }
            }
        }
        if (!flag)
        {
            (*pointsForLocus).push_back((*intersectionsPerpendiculars)[f]);
        }
        flag = false;
    }
}

void sort(vector <Point>* const pointsForLocus, Point point)
{
    vector <pair <double, Point>> vec;

    for (int j = 0; j < (*pointsForLocus).size(); j++)
    {
        double degree;
        Line AB; //Прямая, соединяющая точки
        if (point.x != (*pointsForLocus)[j].x && point.y !=
(*pointsForLocus)[j].y)
        {
            AB.k = (point.y - (*pointsForLocus)[j].y) / (point.x -
(*pointsForLocus)[j].x);

            degree = (atan(AB.k) * 180 / PI); //От -90 до 90 градусов

            //Сейчас определяем правильную четверть

            if (degree > 0) //Значит сейчас он от 0 до 90 градусов
            {
```

```
                if (point.x > (*pointsForLocus)[j].x) //Значит нужно к углу
прибавить 180 градусов
                {
                    degree += 180;
                }
            }
            else //Значит сейчас он от -90 до 0 градусов
            {
                if (point.x > (*pointsForLocus)[j].x) //Значит нужно к углу
прибавить 180 градусов
                {
                    degree += 180;
                }
            }
            //В итоге все углы будут от -90 до 270 градусов
        }
        else
        {
            if (point.x == (*pointsForLocus)[j].x)
            {
                AB.x = point.x; //Значит пряммая, соединяющая сайт и
рассматриваемую точку - параллельна OY, x = AB.x
                if (point.y > (*pointsForLocus)[j].y)
                    degree = 270;
                else
                    degree = 90;
            }
            else
            {
                AB.y = point.y; //Значит пряммая, соединяющая сайт и
рассматриваемую точку - параллельна OX, y = AB.y
                if (point.x > (*pointsForLocus)[j].x)
                    degree = 180;
                else
                    degree = 0;
            }
        }
        vec.push_back(make_pair(degree, (*pointsForLocus)[j]));
    }
    (*pointsForLocus).clear();

    sort(vec.begin(), vec.end());

    for (auto p : vec)
    {
        (*pointsForLocus).push_back(p.second);
    }
}

vector <vector <Point>> determineLocuses(vector <Point> points)
{
    vector <Line> medianPerpendiculars;
    vector <Point> intersectionsPerpendiculars;
    vector <Point> pointsForLocus;

    vector <vector <Point>> locuses;

    for (int i = 0; i < points.size(); i++)
    {
        //Получение (n-1) серединных перпендикуляров для данного сайта
        medianPerpendicularsForSite(points[i], points,
&medianPerpendiculars);
```

```
        //Получение точек пересечения серединных перпендикуляров между собой
и с осями координат
        intersectionsPerpendicularsForSite(&medianPerpendiculars,
&intersectionsPerpendiculars);

        //Сейчас у нас есть все точки пересечения серединных перпендикуляров
в векторе intersectionsPerpendiculars
        //Сайт points[i]
        //Проверка точек на пренадлежность всем полуплоскостям - для
выявления точек для локуса
        pointsForLocusOfSite(&medianPerpendiculars,
&intersectionsPerpendiculars, &pointsForLocus, points[i]);

        //Функция сортировки точек для локуса, в нужном для построения локуса
порядке
        sort(&pointsForLocus, points[i]);


        //Добавляем вектор точек локуса, готового к построению, в общий
список локусов
        locuses.push_back(pointsForLocus);

        medianPerpendiculars.clear();
        intersectionsPerpendiculars.clear();
        pointsForLocus.clear();
    }
    return locuses;
}

int main()
{
    setlocale(LC_ALL, "rus");
    srand(3);
    int n = rand() % 10 + 2; //количество заданных точек (сайтов)

    vector <Point> points;

    for (int i = 0; i < n; i++)
    {
        double x = rand() % 9 + 1, y = rand() % 9 + 1;
        points.push_back(Point(x, y));
    }

    removeRepetitions(&points);
    n = points.size();

    cout << "Выводим нагенерированные точки" << endl;
    for (int i = 0; i < n; i++)
    {
        cout << points[i].x << " " << points[i].y << endl;
    }

    vector <vector <Point>> locuses = determineLocuses(points);

    sf::RenderWindow window(sf::VideoMode(1000, 1000), "Diagram");
    while (window.isOpen())
    {
        sf::Event event;
        while (window.pollEvent(event))
        {
            if (event.type == sf::Event::Closed)
                window.close();
        }
        window.clear();
```

```
        sf::ConvexShape conv;

        int r = 25, g = 20, b = 30;

        for (int i = 0; i < locuses.size(); i++)
        {
            // resize it to i points
            conv.setPointCount(locuses[i].size());

            // define the points
            for (int j = 0; j < locuses[i].size(); j++)
            {
                conv.setPoint(j, sf::Vector2f(locuses[i][j].x * 100,
locuses[i][j].y * 100));
            }

            // set the color
            conv.setFillColor(sf::Color(r, g, b));

            window.draw(conv);
            r += 15;
            g += 10;
            b += 10;
        }
        // define a circle with radius = 5
        sf::CircleShape circle(5.f);

        for (int i = 0; i < points.size(); i++)
        {
            circle.setPosition(points[i].x * 100, points[i].y * 100);

            circle.setFillColor(sf::Color::White);

            window.draw(circle);
        }

        window.display();
    }

}
```