МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Кубанский государственный университет»

**«Алгоритмы на плоскости»**

Курсовая работа

Выполнил:

Пасько Дмитрий Анатольевич,

ФМиКН 2 курс группа 22.2

Краснодар 2017

Содержание:

[Постановка задачи 3](#_Toc482380007)

[Анализ. Описание главных действующих алгоритмов 4](#_Toc482380008)

[Проектирование. Общая структура системы 5](#_Toc482380009)

[Чтение данных и работа с данными 5](#_Toc482380010)

[Поиск решения 6](#_Toc482380011)

[Иллюстрация решения 7](#_Toc482380012)

[Псевдокод программы 7](#_Toc482380013)

[Сложность алгоритма 8](#_Toc482380014)

[Реализация 10](#_Toc482380015)

[Код программы 10](#_Toc482380016)

[Ошибки, с которыми я столкнулся при отладке первых версий программы: 15](#_Toc482380017)

[Тестовые примеры 16](#_Toc482380018)

# Постановка задачи

Создание программы начинается с постановки задачи. Понимание этой задачи является ключевым моментом программы. Исходная **задача** звучит так: *построить два треугольника с вершинами в заданном множестве точек на плоскости так, чтобы первый треугольник лежал строго внутри второго*. Иными словами, задаётся множество точек плоскости и требуется найти и построить два таких треугольника с вершинами в каких-то точках заданного множества, чтобы все точки второго треугольника принадлежали внутренности первого.

**Чтение** *мощности множества* точек плоскости и *элементов множества* (то есть координат точек на плоскости) осуществляется из файла input.txt. В другой файл output.txt программой осуществляется **вывод** координат вершин двух найденных треугольников или сообщение о том, что искомые треугольники построить невозможно. В третий output.bmp файл **выводится изображение** найденных треугольников, если они найдены; в противном случае этот файл не затронется. Код программы пишется на языке С++.

**Основная сложность алгоритма** заключается в построении множества всех треугольников, над которыми будет вестись работа, ибо для этого придётся провести *сортировку исходных точек*, а также *составлять потенциальные треугольники таким образом, чтобы в целях ускорения программы исключить повторения*.

# Анализ. Описание главных действующих алгоритмов

Программа состоит из следующих **этапов**:

1. **Чтение данных и работа с этими данными**. На этом этапе считываются мощность множества (точек плоскости) и координаты точек плоскости, затем проверяется соответствие количества ***разных*** точек мощности множества; в случае несоответствия программа говорит о невозможности решения задачи, а в случае соответствия переходит к следующему этапу, где перебирается множество всех ***различных*** треугольников с вершинами в заданных точках, над которыми будет производиться анализ на следующем этапе.
2. **Поиск решения задачи**. Путём перебора всех различных пар разных треугольников ведётся поиск двух таких, чтобы один лежал строго внутри другого. Поиск ведётся до тех пор, пока пара треугольников с такими свойствами не будет найдена или все пары не будут перебраны; в первом случае задача считается решённой и осуществляется переход к иллюстрированию, а во втором случае выносится сообщение о невозможности решения задачи.

**Иллюстрация решения в случае его существования**. При помощи графических возможностей языка С++ найденные треугольники (если их удалось найти) изображаются на рисунке.

Далее проводится подробное описание каждого из этапов.

# Проектирование. Общая структура системы

## Чтение данных и работа с данными

**Чтение**. С текстового файла **in.txt** считываются пустая строка и все числовые значения, которые в нём есть; количество числовых значений **К** считается, а результатом целочисленного деления **К**/2 выясняется мощность **М** множества точек плоскости, то есть количество задаваемых точек; далее эти точки снова считываются и сохраняются в массив; раз по условию задачи один треугольник должен лежать строго внутри второго, то эти треугольники не могут иметь общих вершин и поэтому мощность множества, очевидно, не должна быть меньше шести (по три вершины у каждого треугольника); если **М** < 6, программа выдаст соответствующее сообщение; важно заранее проверить условие на мощность множества, чтобы избежать лишних и заведомо бесполезных вычислений. В программе точки являются *структурами* ***Point*** *с полями* ***x*** *и* ***y*** (координаты точки), причём ***x***, ***y*** – переменные *с плавающей точкой* (типа **double**); а массив точек является *массивом структур* размерности [**М**].

Далее производится **проверка** **на совпадение введённых точек** с той же целью: различных точек должно быть не менее шести; и для большого числа точек гораздо выгоднее отсортировать массив и отсеять повторяющиеся точки, нежели «пускать» их в основной алгоритм, нагромождая программу лишними вычислениями. Алгоритм этой проверки таков: точки сортируются по первой координате, потом полученный массив разбивается на участки, где имеются точки с одинаковой первой координатой, далее каждый такой участок сортируется по второй координате, а затем все точки пробегаются; если у соседних точек обе координаты равны, одна из точек удаляется, а пробег продолжается; если после пробега точек становится меньше шести, выведется сообщение, а в противном случае работа продолжается. В конечном итоге получается окончательный массив точек плоскости; если мощность исходного множества уменьшится, это зафиксируется. Необходимо пояснить, что сортировку неизбежно придётся проделать в два этапа, чтобы точки с одинаковыми координатами в нём оказались рядом, на соседних местах; для сортировки используется библиотечная функция sort().

Из оставшихся точек теоретически **может быть сформирован массив треугольников**; все треугольники должны быть разными, то есть иметь разные вершины, иначе придётся по шесть раз проверять фактически один и тот же треугольник, что в десятки раз удлинит время работы программы. Согласно комбинаторике, число **Т** таких треугольников равно , где **M** – теперь мощность множества точек после проверки на совпадение; для удобства будем считать треугольником массив из трёх структур (трёх его вершин) в теоретическом массиве структур размерности [3**T**]. Сложность заключается в том, чтобы расположить в этот массив все тройки точек, ***образующих треугольник*** (то есть следует следить за неравенством треугольника), так, чтобы они не повторялись; осуществляется это по следующему алгоритму: в массиве точек размерности [**М**] берётся точка (1) – первая, затем фиксируется точка (2), а точка (3) пробегает по оставшимся; после этого точка (2) переходит в точку (3), а третья пробегает по следующим, и цикл повторяется до тех пор, пока вторая точка не перейдёт в (**М** – 1)-ю; после этого точка (1) перейдёт в точку (2), повторится весь цикл, а потом первая точка снова осуществит переход; закончится общий цикл тогда, когда первая точка перейдёт в точку (**М** – 2); параллельно для всякой найденной тройки точек будет проверяться неравенство треугольника, так что потенциальное количество треугольников может уменьшиться. Но чтобы не задействовать много оперативной памяти и много времени (ведь важно найти любые два треугольника, обладающие известным свойством по отношению друг к другу), решено алгоритм перебора осуществлять уже в самом цикле поиска, дабы можно было прервать его хоть в самом начале, если решения уже будут найдены.

## Поиск решения

По алгоритму, аналогичному предыдущему (но только теперь с двумя «точками»), **составляются пары треугольников**, причём ни одна пара не должна совпасть с ранее найденной (*если первый треугольник уже зафиксирован, то второй будет составляться для первых найденных трёх точек, расположенных внутри первого*). Параллельно для каждой пары треугольников (то есть для вершин второго треугольника) из массива треугольников **проверяется названное свойство**: один треугольник находится строго внутри другого. Для оптимизации вычислений проверка происходит в три этапа:

1. **Необходимость** для точки – *расстояние от каждой вершины потенциально внутреннего треугольника до всякой вершины потенциально внешнего треугольника должно быть меньше длины наибольшей стороны внешнего треугольника*; очевидно, что каждая точка внутри треугольника обладает таким свойством, а сильно удалённые от треугольника точки (вне описанной окружности) – не обладают; доказательство очевидно. Для проверки необходимого условия требуется найти длины сторон «внешнего» треугольника и соответствующие расстояния, а эти данные потребуются и для достаточности. Тесты показали, что в среднем по несоответствию необходимому условию отсеиваются 70% точек.
2. **Достаточность** для точки – *каждая вершина потенциально внутреннего треугольника составляет с вершинами потенциально внешнего треугольника ровно три маленьких треугольника* (если хотя бы для одного не будет выполнено неравенство треугольника, пара меняется), *сумма площадей которых равна площади «внешнего» треугольника* (), конечно, с учётом погрешности; площадь треугольника считается по известной формуле Герона: , все данные для которой найдены ранее.
3. **Найденные точки должны составлять треугольник.** Все найденные внутренние точки треугольника будут записаны в отдельный массив, а потом из массива будут найдены такие 3 точки, которые составляют внутренний треугольник. Если внутренний треугольник составить не удастся, меняется внешний и цикл повторяется.

Если искомые решения не будут найдены, в файл вывода будет записано соответствующее сообщение. В противном случае **в файл выводятся координаты вершин найденных треугольников** и начинается следующий этап программы.

Однако, *для ускорения работы программы решено проводить проверку уже в процессе построения массива треугольников; это позволит не только решить задачу быстрее, но и не потребует много динамической памяти*. Таким образом, поиск решение проводится уже при составлении треугольников, а также производится дополнительное ограничение в том, чтобы *треугольники с одной общей вершиной не проверялись, если уже известно, что эта вершина находится вне потенциально внешнего треугольника*.

## Иллюстрация решения

С помощью библиотеки “Graph.h” происходит построение указанных треугольников, если они найдены. Если два треугольника найдены, то их вершины будут располагаться в своих массивах и иметь координаты ; следовательно, отрезком, соединяющим точки , где , будет кусок прямой или ; однако, библиотека “Graph.h” располагает функцией, которая рисует прямые между точками, поэтому иллюстрация решения не будет составлять труда.

## Псевдокод программы

Покажем алгоритм программы нагляднее в следующем псевдокоде:

//основная функция

Функция main()

{

**чтение данных и работа с данными()**{}

**поиск решения()**{}

}

//внутренние функции и процедуры основной функции

**Чтение данных и работа с данными**()

{

чтение мощности множества;

достаточно ли точек, чтобы составить два треугольника? Иначе соответствующий вывод и прекращение работы программы;

чтение точек плоскости и заполнение массива этих точек;

сортировка массива точек;

удаление из массива повторяющихся точек;

достаточно ли осталось точек, чтобы составить два треугольника? Иначе соответствующий вывод и прекращение работы программы;

}

**Поиск решения**()

{

Цикл перебора всевозможных троек точек плоскости (вершин внешнего треугольника)

{

Если тройка составляет треугольник

{

Цикл перебора всевозможных точек плоскости (вершин внутреннего треугольника)

{

Если необходимое условие не выполняется, перейти к следующей точке;

Если достаточное условие не выполняется, перейти к следующей точек;

Внести точку в массив внутренних точек данного внешнего треугольника;

}

Если в массиве внутренних точек есть такие, которые составляют треугольник, то решение найдено, решение выводится, производится **иллюстрация решения**, программа прекращает работу;

}

}

если программа дошла до этого места, то задачу решить не удалось;

}

## Сложность алгоритма

*Сложность* этого алгоритма оценить крайне трудно, поскольку программа сильно зависит от входных значений: она может закончить работу, как только узнает о недостаточном количестве точек или всего через несколько действий, если искомые точки найдутся быстро. Возможно, описанный алгоритм не является наиболее оптимальным, но, по всей видимости, усилия на ещё большее совершенствование алгоритма того не стоят, поскольку специфика задачи предполагает, что код не будет использоваться часто и, уж тем более, с большим количеством входных точек. Оценим *сложность алгоритма* при самом неблагоприятном «множестве» точек (которое, быть может, и не существует), а именно: 1) количество **M** точек очень велико; 2) среди точек нет повторяющихся; 3) любые три «внешние» точки образуют треугольник; 4) для любых трёх «внешних» точек все прочие лежат внутри треугольника, образованного на этих точках как на вершинах; 5) никакие «внутренние» точки не будут образовывать треугольник; в таком случае сложность алгоритма будет суммой следующих величин:

– сложность алгоритма сортировки точек, где **М** – мощность множества точек

– неоспоримо крупная величина, но имеет она место лишь при указанных пяти условиях. При тестировании программа в среднем работала меньше секунды, если не нужно было иллюстрировать решение, и не больше семи секунд, если это требовалось.

# Реализация

Согласно **Бьярне Страуструпу**,

“Цель программиста – выразить вычисления, причём это должно быть сделано **1)** правильно, **2)** просто, **3)** эффективно”[[1]](#footnote-1).

Именно поэтому код программы снабжён множественными *комментариями*, переменные названы *удобными именами*, а сама программа разбита на большое число отдельных *функций* и *процедур*, чтобы легче было проводить её отладку, и читать, и понимать. Если общий алгоритм программы требовал пояснений, то отдельные функции в них не нуждаются, поэтому код программы можно предоставить уже сейчас:

## Код программы

//программа, которая из дискретного множества точек находит такие две тройки, что треугольники, образованные этими тройками точек как вершинами, являются друг для друга внешним и внутренним соответственно;

//программа иллюстрирует решение, если оно найдено

//написал ПАСЬКО Д. А.

//в последний раз программа редактировалась 12.05.2017

//сложность может вызвать функция search()

//при организации кода я пытался разбить программу на как можно большее число независимых фрагментов

//программа считается готовой

//директивы препроцессора

#include <iostream>

#include <fstream>

#include <algorithm>

#include <cstdlib>

#include <cmath>

#include "Graph.h"

//стандартное пространство имён

using namespace std;

struct Point {//структрура ТОЧКА на плоскости

double x;//поле - координата по х

double y;

};

//глобальные переменные

int m;//объявление мощности множества точек плоскости

int kt;//объявление мощности множества РАЗЛИЧНЫХ точек плоскости, то есть множества точек после отсеивания повторяющихся

Point z1[3], z2[3]; //два маленьких массива, в которых будут располагаться вершины найденных треугольников

Point \*mas;//указатель на массив точек плоскости

Point \*internalpoints;//указатель на массив "внутренних точек"

double a0, b0, c0, d0;//пределы окна

const double eps = 0.01;//погрешность, необходимая в некоторых местах программы

const int nRis = 250;//количество шагов (нужно при иллюстрировании)

//перечисление прототипов вспомогательных функций

void building();

void search();

void illustrating();

void read();

void screening();

double P(double a, double b, double c);

void write1();

void write2();

void exceptionmas(Point \*mas);

void excep(int i, Point \*a);

void Ris(Point \*z, int i, int j);

double min3(double a, double b, double c);

double max3(double a, double b, double c);

double eudistance(Point a, Point b);

bool triangleanswer(Point mas1, Point mas2, Point mas3);

bool triangleanswer(Point \*a, int k);

bool comp(Point a, Point b);

//основная функция

int main()

{

building();//чтение и работа с данными

search();//поиск решения

//окончание программы

return 0;

}

//тела вспомогательных функций для main (основной функции)

void building()

{

read();//чтение данных из файла

screening();//отсеивание из массива точек одинаковых точек

}

void search()

{

internalpoints = new Point[kt - 3];//массив "внутренних точек", в который для каждого потенциально внешнего треугольника будут записываться внутренние точки, ибо не любая тройка из них будет составлять треугольник, так что нужна проверка

for (int t11 = 0; t11 < (kt - 2); t11++)//цикл поиска точек первого треугольника

{

for (int t12 = t11 + 1; t12 < (kt - 1); t12++)

for (int t13 = t12 + 1; t13 < (kt); t13++)

{

double a, b, c;//объявление сторон треугольника

a = eudistance(mas[t11], mas[t12]);//расстояние между точками t11 и t12

b = eudistance(mas[t13], mas[t12]);//расстояние между точками t13 и t12

c = eudistance(mas[t11], mas[t13]);//расстояние между точками t11 и t13

if (triangleanswer(mas[t11], mas[t12], mas[t13]))//проверка, составляют ли точки треугольник

{

z1[0] = mas[t11];

z1[1] = mas[t12];

z1[2] = mas[t13]; //фиксация точек внешнего треугольника в его массив

double p1, p2, p3;//объявление расстояний до найденных вершин

int k = 0; //индекс, равный количеству найденных точек внутри треугольника

for (int s = 0; s < kt; s++)

{

if (triangleanswer(mas[s], z1[1], z1[2]) && triangleanswer(z1[0], mas[s], z1[2]) && triangleanswer(z1[0], z1[1], mas[s]))//если точка образует треугольники с вершинами уже найденного треугольника (не лежит на его сторонах в том числе)

{

p1 = eudistance(z1[0], mas[s]);//расстояние между точками t11 и s

p2 = eudistance(z1[1], mas[s]);//расстояние между точками t12 и s

p3 = eudistance(z1[2], mas[s]);//расстояние между точками t13 и s

if (max3(p1, p2, p3) >= max3(a, b, c)) continue; //проверка необходимого условия

if (P(a, p1, p2) + P(b, p2, p3) + P(c, p1, p3) >= P(a, b, c) + eps) continue;//проверка достаточного условия (неравенство площадей Р)

k++;//одна из точек внутреннего треугольника найдена, увеличивается счётчик

internalpoints[k - 1] = mas[s];//фиксация найденной точки

}

}

if (k >= 3)//если внутренних точек хотя бы 3

{

if (triangleanswer(internalpoints, k))//если в массиве внутренних точек существует тройка, образующая треугольник

{

write1();//вывод точек

//переход к иллюстрированию, ибо решение найдено

illustrating();//иллюстрирование решения

return;//прекращение работы программы (завершение функции поиска)

}

}

}

}

}

write2();//если, дойдя до этого момента, программа не нашла решения, сообщается о том, что решения не существует

return;//...и программа прекращает работу

}

//тела вспомогательных функций следующего порядка

//вспомогательные для "чтения данных и работы с данными"

void read()

{

char buff[150];//создание вспомогательного символьного массива

int k = 0; //счётчик

double r;//вспомогательное действительное число

ifstream fin("input.txt"); //объявление объекта для чтения из файла

fin >> buff;//чтение ненужной текстовой информации

while (fin >> r) k++;//пока координаты точек считываются, прибавлять к счётчику

m = k / 2;//вычисление мощности множества

if (m < 6) { ofstream fout("output.txt"); fout << "точек слишком мало"; fout.close(); exit(0); }//условие на мощность множества

mas = new Point[m];//создание массива точек

fin.clear();

fin.seekg(0, ios::beg);//поставить указатель на начало файла, чтобы считать значения заново

fin >> buff;

for (int i = 0; i < m; i++) fin >> mas[i].x >> mas[i].y;//заполнение массива точек

fin.close();

}

void screening()

{

sort(mas, mas + m, comp);//сортировка точек по компаратору

exceptionmas(mas);//отсеивание повторяющихся точек

if (kt < 6) { ofstream fout("output.txt"); fout << "разных точек слишком мало"; fout.close(); exit(0); }//условие на мощность множества

}

bool comp(Point a, Point b)//функция компоратора

{

if (a.y < b.y)return true;//cравнение по второй координате

else if (a.y > b.y) return false;

else return a.x < b.x;//если вторые координаты равны, сравнение по первой координате

}

void exceptionmas(Point \*mas)//отсеивание из массива повторяющихся элементов

{

kt = m;

for (int i = 1; i < kt; i++)

{

if (mas[i - 1].x == mas[i].x && mas[i - 1].y == mas[i].y) { excep(i, mas); kt--; i--; }//удаление из массива точек повторяющихся элементов

}

}

void excep(int i, Point \*a)//удаление из массива элемента i

{

for (int j = i; j < kt - 1; j++) a[j] = a[j + 1];

}

//вспомогательные для "поиска"

double P(double a, double b, double c)

{

double p, s;

p = (a + b + c) / 2;//полупериметр

s = sqrt(p\*(p - a)\*(p - b)\*(p - c));//площадь

return s;

}

void write1()

{

ofstream fout("output.txt");

fout << "вершины первого треугольника" << endl;

for (int i = 0; i < 3; i++)fout << z1[i].x << " " << z1[i].y << endl;

fout << "вершины второго треугольника" << endl;

for (int i = 0; i < 3; i++)fout << z2[i].x << " " << z2[i].y << endl;

fout.close();

}

void write2()

{

ofstream fout("output.txt"); fout << "решения не существует"; fout.close();

}

double min3(double a, double b, double c)//минимум из тройки элементов

{

return min(min(a, b), c);

}

double max3(double a, double b, double c)//максимум из тройки элементов

{

return max(max(a, b), c);

}

double eudistance(Point a, Point b)//евклидово расстояние между точками

{

return sqrt(pow((a.x - b.x), 2) + pow((a.y - b.y), 2));

}

bool triangleanswer(Point mas1, Point mas2, Point mas3)

{

double a, b, c;//объявление сторон треугольника

a = eudistance(mas1, mas2);//расстояние между точками mas1 и mas2

b = eudistance(mas3, mas2);//расстояние между точками mas3 и mas2

c = eudistance(mas1, mas3);//расстояние между точками mas1 и mas3

if ((a + b > c) && (a + c > b) && (c + b > a))//проверка неравенств треугольника

return true;

else return false;

}

bool triangleanswer(Point \*a, int k)

{

for (int t1 = 0; t1 < (k - 2); t1++)//цикл поиска точек треугольника

for (int t2 = t1 + 1; t2 < (k - 1); t2++)

for (int t3 = t2 + 1; t3 < k; t3++)

{

if (triangleanswer(a[t1], a[t2], a[t3]))//если точки составляют треугольник, то они сразу записываются в массив точек внутреннего треугольника

{

z2[0] = a[t1];

z2[1] = a[t2];

z2[2] = a[t3];

return true;

}

}

return false;

}

//иллюстрирование

void illustrating()

{

//инициализация пределов окна

a0 = -1 + min3(z1[0].x, z1[2].x, z1[1].x);

b0 = 1 + max3(z1[0].x, z1[2].x, z1[1].x);

c0 = -1 + min3(z1[0].y, z1[2].y, z1[1].y);

d0 = 1 + max3(z1[0].y, z1[2].y, z1[1].y);

SetColor(250, 250, 250); // задаем фоновый цвет окна (белый)

SetWindow(a0, b0, c0, d0); // создаём окно (создаем массивы R,G,B)

// с пределами [a,b]x[c,d]

SetColor(0, 0, 0); // задаем цвет координатных осей (чёрный)

xyLine(0, 0, 1, 1); // строим оси, пересекающиеся в т. (0,0), с шагом делений по х равным 1 и 1 по у

// рисуем график

Ris(z1, 1, 2); //отрезок между первой и второй вершиной первого треугольника

Ris(z1, 3, 2);//отрезок между третьей и второй вершиной первого треугольника

Ris(z1, 1, 3);//отрезок между первой и третьей вершиной первого треугольника

Ris(z2, 1, 2);

Ris(z2, 3, 2);

Ris(z2, 1, 3);//отрезок между первой и третьей вершиной второго треугольника

CloseWindow();// закрываем окно (создаем bmp-файл)

}

void Ris(Point \*z, int ii, int jj)//изображение отрезка между точками zii и zjj

{

SetColor(0, 255, 0); // задаем цвет линии - зелёный

SetPoint(z[ii - 1].x, z[ii - 1].y); // устанавливаем курсор на точку zii

Line2(z[jj - 1].x, z[jj - 1].y);// строим отрезок,соединяющий курсор с новой точкой zjj, и перемещаем курсор в эту точку

}

## Ошибки, с которыми я столкнулся при отладке первых версий программы:

1. Программа верно считывала координаты точек при первом прочтении файла, но почти ничего не считывала при втором прочтении.
2. Сортировка массива структур через функцию **sort** стандартной библиотеки не заработала. Пришлось долго думать над ней, но в итоге всё получилось и код программы стал короче.
3. При отсеивании исчезали почти все точки.
4. Программа не находила решение даже там, где оно имелось. Проблема заключалась в неверном присваивании.
5. Программа не находила решения, потому что не видела никаких треугольников (не тот знак поставил в неравенстве треугольника).
6. Программа не находила всех внутренних точек (возникла проблема в достаточном признаке).
7. Программа не считывала числа с плавающей точкой.
8. Вместо отрезков программа рисовала какие-то колебания, поскольку произошло совпадение имён.

## Тестовые примеры

1. Мощность множества получается меньше 6-ти. **Ожидаемый ответ**: точек слишком мало.
   1. 1 2

4 5

5 6

7 8

**ОТВЕТ**: точек слишком мало

* 1. 2 3

0 0

8 9

7 6

5

**ОТВЕТ**: точек слишком мало

1. Количество *разных* введённых точек меньше 6-ти. **Ожидаемый ответ**: разных точек слишком мало.

1 2

2 1

4 5

6 7

9 8

1 2

**ОТВЕТ**: разных точек слишком мало

1. Точки располагаются на одной прямой, так что треугольники вообще не получаются. **Ожидаемый ответ**: решения не существует.

1 1

2 2

3 3

4 4

5 5

6 6

7 7

8 8

**ОТВЕТ**: решения не существует

1. Точки располагаются так, что ни в одном потенциально внешнем треугольнике не найдутся три внутренние точки. **Ожидаемый ответ**: решения не существует.

0 0

4 0

2 4

2 1

2 2

5 3

**ОТВЕТ**: решения не существует

1. Точки расположены так, что даже если в потенциально внешнем треугольнике найдутся три внутренние точки, то внутреннего треугольника не получится, ибо внутренние точки лежат на одной прямой, не образуя треугольник. **Ожидаемый ответ**: решения не существует.

0 0

3 −6

3 3

2 1

2 0

2 −1

2 −2

**ОТВЕТ**: решения не существует

1. Точки располагаются так, что нельзя построить один треугольник строго внутри другого, но можно построить такой треугольник, что одна или более его вершин будут на сторонах внешнего треугольника. **Ожидаемый ответ**: решения не существует.

0 0

3 0

0 5

1 0

0 1

1 2

**ОТВЕТ**: решения не существует

1. Точки расположены так, что решение существует. **Программа должна** вывести координаты решения и нарисовать его.

0 0

4 −2

5 3

4 1

3 1

4 2

**ОТВЕТ**:

вершины первого треугольника

4 -2

0 0

5 3

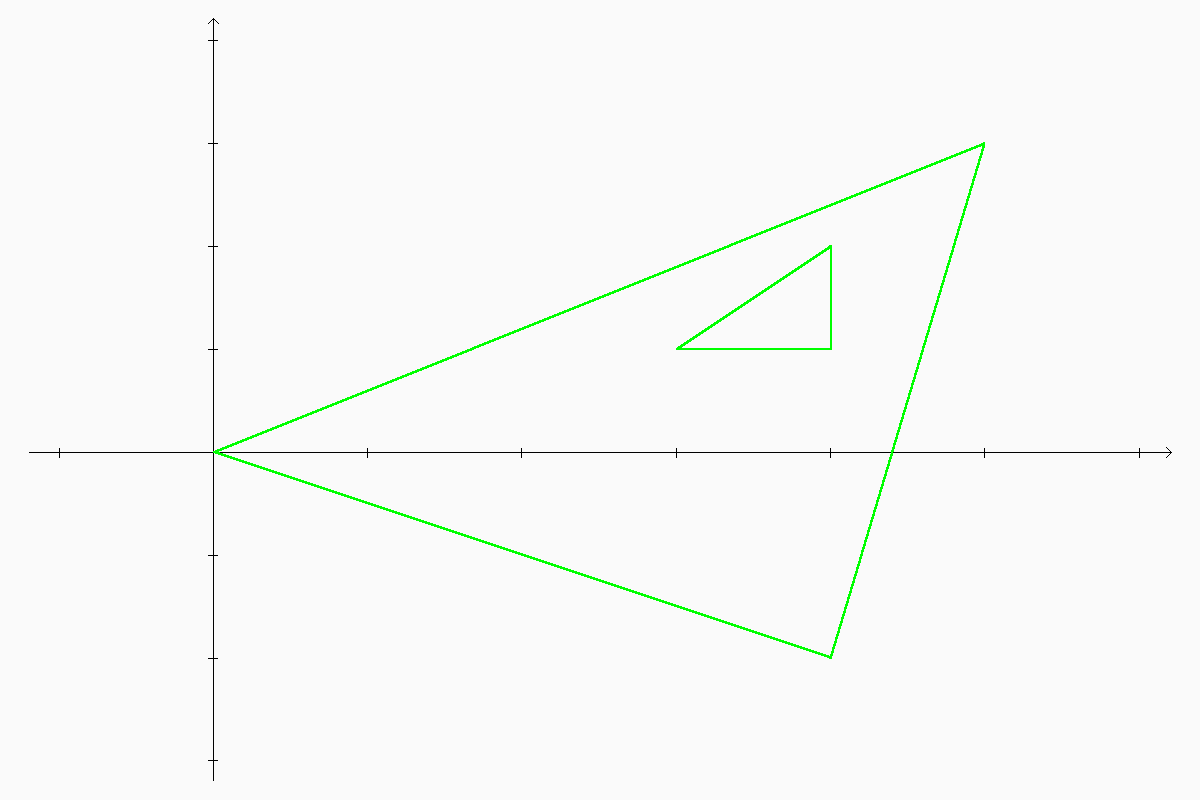
вершины второго треугольника

3 1

4 1

4 2

**РИСУНОК**:



1. Другой пример того, когда решение существует. **Программа должна** вывести координаты решения и нарисовать его.

0 0

11 0

12.2 0

6 1

8 2

4 2

5 3

7.5 3

6 5

6 9

0 7

14 7

4.5 −2

**ОТВЕТ**:

вершины первого треугольника

4.5 -2

0 0

14 7

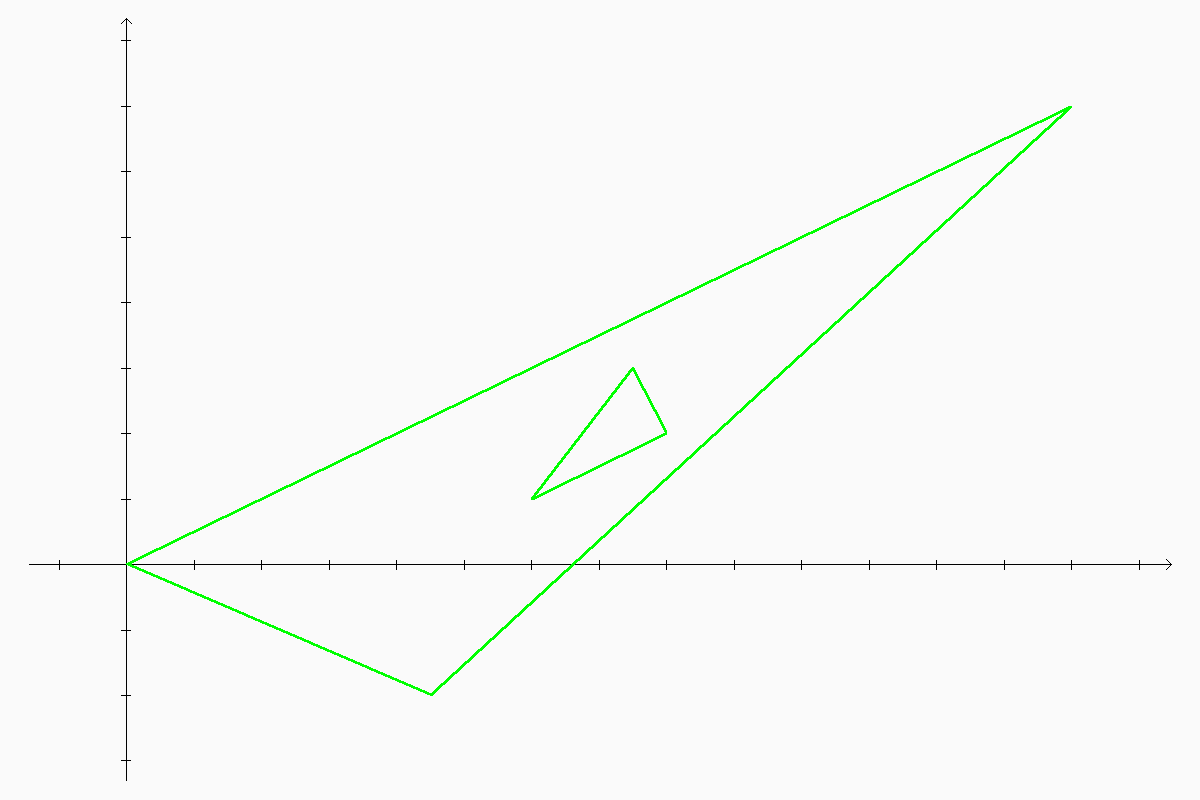
вершины второго треугольника

6 1

8 2

7.5 3

**РИСУНОК**:



1. В заключение возьмём *произвольный крупный набор* точек.

1 1

2 1

2 2

3 4

5 6

7 8

0 11

45 36

67 89

−10 2

1 −3

−14 −16

3 4

5 6

7 8

0 11

45 36

1 1

2 1

2 2

3 4

5 6

7 8

0 11

45 36

67 89

−10 2

1 −3

−14 −16

3 4

5 6

7 8

0 11

45 36

4 2

5 3

7.5 3

6 5

6 9

0 7

14 7

4.5 −2

**ОТВЕТ**:

вершины первого треугольника

-14 -16

1 -3

3 4

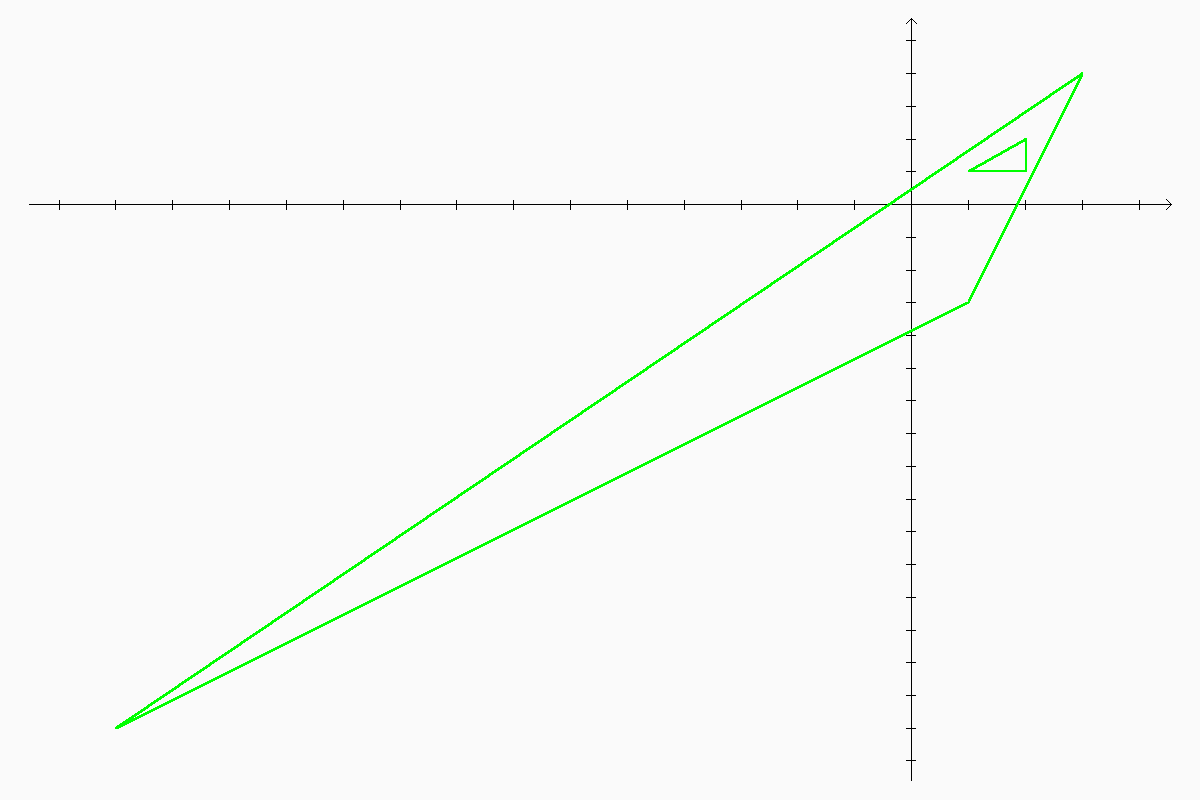
вершины второго треугольника

1 1

2 1

2 2

**РИСУНОК**:



1. «Программирование: принципы и практика с использованием С++» [↑](#footnote-ref-1)