**ГБОУ "Президентский ФМЛ № 239"**

**Plottery**

Нахождение такой прямой, проходящей через две точки множества, что количество точек слева и справа от нее является минимально возможным.

**Работу выполнил:**

Ученик 11-7 класса

Луняк Николай

**Учитель информатики:**

Батракова Людмила Васильевна

**Учитель геометрии:**

Аксенов Константин Николаевич

**1 Постановка задачи**

*Текст задания:* нахождение такой прямой, проходящей через две точки множества, что количество точек слева и справа от нее является минимально возможным.

*Интерпретация:* имеется множество точек. Необходимо найти прямую, проходящую через 2 точки, и количество точек слева и справа от нее должно быть минимальным (в идеальном случае равным).

**2 Входные данные**

В качестве входных данных выступает множество точек с действительными координатами. Для конкретности координатная сеть ограничена до диапазона [-128;128] по осям OX и OY. Соответственно эти границы предполагаются и для координат точек. Количество точек было решено ограничить до 300 шт. на 1 слой.

Информация о точках может быть введена посредством клавиатуры, мыши, генератора случайностей, файла и линейной и квадратичной функцией.

**3 Выходные данные**

Под выходными данными следует подразумевать прямую, удовлетворяющую условию. Она отражается напрямую на координатной сетке, а также может быть сохранена в файл вместе со всеми остальными объектами сцены (прямыми и точками).

**4 Математическая модель**

Для точек и :

Где – это тангенс угла наклона α прямой, пересекающей точки и , а – это функция этой прямой.

*α*

*x2*

*x1*

*f(x)*

*P1*

*P2*

*Y*

*X*

*y2*

Найдя *k*, мы можем найти *b:*

*y1*

Для проверки расположения точки относительно прямой достаточно проверить:

Это условия нахождения точки выше прямой. *p* – это определенная погрешность. Визуально достаточно трудно судить однозначно о положении точки. Поэтому для более удобного представления информации было решено определить точность (погрешность смещения точки) – такую величину, при отклонении точки на Δy от прямой, меньшем которой точка считается принадлежащей этой прямой. Условие нахождения точки ниже прямой:

*f(x)*

*p*

*yi*

*f(xi)*

*Pi*

*xi*

*X*

*Y*

**5 Анализ входных данных**

В целях потенциальной возможности расширения функционала в будущем было решено представить активный сеанс работы с приложением как работу на «Сцене», «Проект» которой может быть сохранен в файл. Сцена включает в себя список слоев. Каждый слой в свою очередь содержит в себе список точек с максимальным ограничением до 300 шт. (формально, так как список по своей природе не ограничен в размере элементов) и список отрезков, которые могут соединять эти точки между собой.

Одна точка может быть представлена классом **Point** из модуля **uPoint** пакета **LKPackage** (далее **uPoint::Point**)

type Point = class(TObject)

x, y: real;

constructor Create(x, y: real);

end;

Учитывая, что это класс, а в Delphi 5 отсутствует должная сериализация (насколько это мне известно), были созданы еще записи-двойники. При сохранении/чтении файла вся информация переводится сначала в эти записи, и они уже записываются. Так аналогом **Point** в мире записей стала **PointRecord**.

К сожалению, классы Delphi 5 не имеют единого для всех суперкласса. Ближайшим аналогом послужил **TObject.** Вследствие этого сначала был создан **uArrayList::ArrayList**, определенный над элементами-наследниками **TObject**:

type List = Array of TObject;

{

Automatic dynamic array

}

type ArrayList = class

protected

arr: List;

len: integer;

public

constructor Create(); overload;

constructor Create(amount: integer); overload;

function enlargeBy(arr: List; amount: integer): List;

function enlarge(arr: List): List;

procedure add(obj: TObject);

procedure put(obj: TObject; index: integer);

function remove(index: integer): TObject; overload;

procedure remove(obj: TObject); overload;

function size(): integer;

function get(index: integer): TObject;

function find(obj: TObject): integer;

end;

Поля **arr** и **len** хранят массив с вакантными местами и количество занятых мест соответственно. Представленный интерфейс **ArrayList**, полагаю, не нуждается в разъяснении. Однако вкратце опишу некоторые методы:

function enlargeBy(arr: List; amount: integer): List;

Этот метод возвращает новый массив, содержащий все компоненты старого, но большего размера. (Этот метод мог быть приватным, однако мои великодушные тараканы решили не жадничать).

function enlarge(arr: List): List;

Увеличивает список в чуть более чем два раза (2n + 1).

function size(): integer;

Возвращает количество элементов в списке.

function find(obj: TObject): integer;

Производит поиск элемента в списке и возвращает его индекс в случае удачи. В противном случае результатом будет -1.

Для удобства хранения точек был определен **uArrayListOfPoint::ArrayListOfPoint**, наследующий **ArrayList**. В нем добавлены методы для быстрого приведения **TObject** к **Point**. Например,

// ArrayListOfPoint

function ArrayListOfPoint.remove(index: integer): Point;

begin

remove := inherited remove(index) as Point;

end;

Метод **remove** удаляет элемент из списка по его индексу и «отдает» в качестве возвращаемого значения.

Аналогичным образом были объявлены также: **ArrayListOfInt**, **ArrayListOfString**, **ArrayListOfLayer** и **ArrayListOfLine**. (Для **ArrayListOfInt** и **ArrayListOfString** были созданы классы-обертки, наследующие **TObject**: **\_Int**, **\_Str**).

**6 Выбор метода решения**

Для решения поставленной задачи планируется перебор всех пар точек. Для каждой пары будет вычислено уравнение прямой. Далее производится перебор всех остальных точек и определение нахождения каждой относительно прямой. Будет вестись 2 переменные-счетчика: количество найденных точек ниже прямой и количество точек, найденных выше прямой. Если их разность по модулю будет меньше оной для предыдущей пары точек, то «запоминаем» нашу пару. После перебора запомненная пара и будет являться по крайней мере одним из возможных ответов. Теперь, зная уравнение прямой будет построен отрезок на интервале [-128, 128] (для единства представления используются отрезки, а не бесконечные прямые). Этот отрезок будет проходить через пару точек, а также будет лежать на искомой прямой, а учитывая масштаб, задачу можно считать решенной.

**7 Листинг решения задачи**

Здесь будет приведен сугубо код решения задачи, так как суммарный код приложения насчитывает около 7000 строк (да, сидел и считал).

// School Task #1

procedure TaskSchool1(points: PointList);

var

// i & j – counters for testing each pair

// l is used for testing each single point and the line

// left & right – amount of points to the left & right

// dif & mindif - |left - right| and the smallest one

// k, b, minK, minB – function coefficients

// d – precision

// yl – actual difference between point y-coordinate

// and function value

i, j, l, left, right, dif, minDif: integer;

k, b, d, yl, minK, minB: real;

begin

d := 0.1;

minK := 0;

minB := 1;

minDif := -1; // will be reset, thats just for debugger

// for each pair

for i := 0 to length(points) - 2 do begin

for j := i + 1 to length(points) - 1 do begin

// default

left := 0;

right := 0;

k := (points[j].y – points[i].y) / (points[j].x – points[i].x);

b := points[i].y - points[i].x \* k;

// now we got f(x) = kx + b. Testing points

for l := 0 to length(points) - 1 do begin

// not the ones from the pair

if (l <> j) and (l <> i) then begin

// analizing

yl := points[l].y - points[l].x \* k - b;

if abs(yl) > d then begin

if yl > 0 then

left := left + 1

else

right := right + 1;

end

end;

end;

// check if we got the best

dif := abs(left - right);

if (i = 0) and (j = 1) or // first

(dif < minDif) then begin

minDif := dif;

minK := k;

minB := b;

end;

end;

end;

// using Scene API to display the result

currentScene.getCurrentLayer().addPoint(Point.Create(

-128, -128 \* minK + minB

));

currentScene.getCurrentLayer().addPoint(Point.Create(

128, 128 \* minK + minB

));

currentScene.getCurrentLayer().addLine(Line.Create(

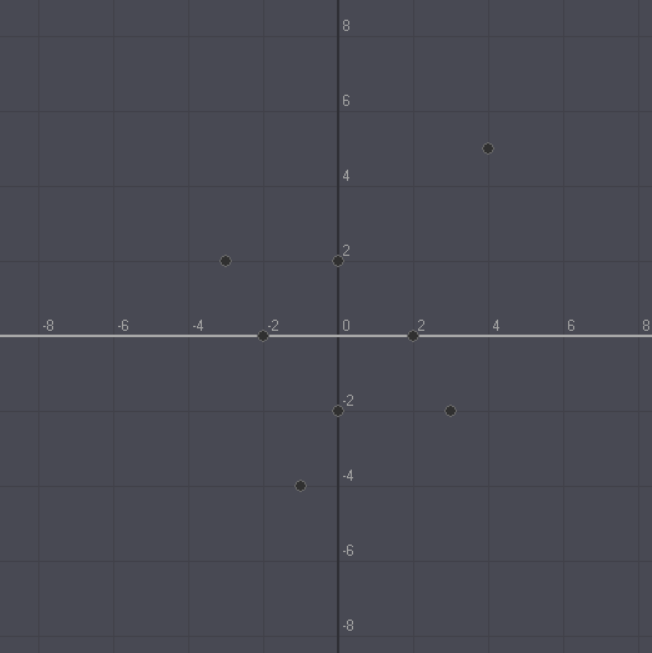
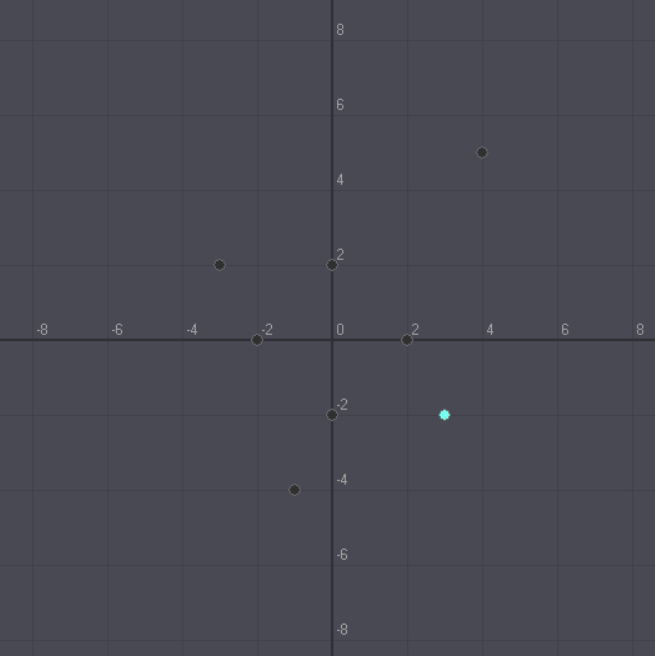
currentScene.getCurrentLayer().currentPoint,

currentScene.getCurrentLayer().currentPoint - 1

));

end;

**8 Примеры работы**

****

**Входные данные**

*P1(2, 0)*

*P2(-2, 0)*

*P3(0, -2)*

*P4(0, 2)*

*P5(4, 5)*

*P6(-3, 2)*

*P7(-1, -4)*

*P8(3, -2)*

**Выходные данные**

*f(x) = 0*

**Входные данные**

*P1(2, 2)*

*P2(-2, -2)*

*P3(-6, 0)*

*P4(-2, 2)*

*P5(-6, 4)*

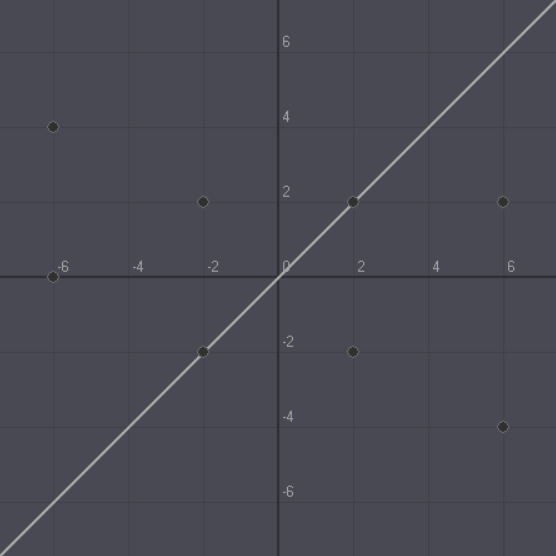
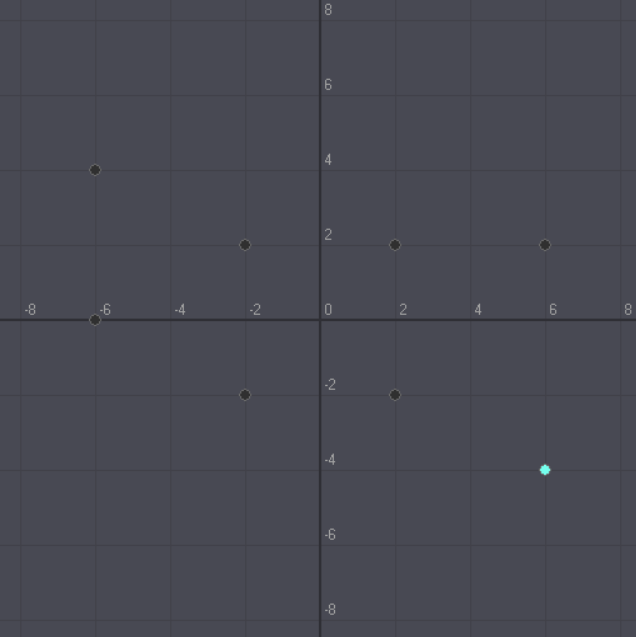
*P6(2, -2)*

*P7(6, 2)*

*P8(6, -4)*

**Выходные данные**

*f(x) = x*

****

**9 Анализ правильности решения**

Так как нужно подсчитать количества точек по обе стороны от функции, то надо определить, что является «стороной». Очевидно, что за стороны можно принять y-координату точки относительно значения функции в ней. Тогда для определения наилучшего совпадения необходимо перебрать все пары точек и найти прямую, проходящую через них, и сравнить ее значение в каждой точке с y-координатой этой точки.