**ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**КАЗАНСКИЙ ПРИВОЛЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**“Институт информационных технологий и**

**интеллектуальных систем”**

**Направление: “Программная инженерия”**

**Семестровая работа № 2**

**по теме:**

**«Поиск ближайшей пары точек методом “разделяй и властвуй”»**

Выполнила: Голдова Е. А.

Преподаватель: Зиятдинов М.Т

Казань 2022

**Краткая историческая справка:**

В начале 1970-х годов этот алгоритм рассматривали М. И. Шамос и Д. Хоуи в ходе проекта по поиску эффективных алгоритмов для базовых вычислительных примитивов для геометрических задач. Эти алгоритмы заложили основу зарождающейся в то время области вычислительной геометрии и проникли в такие области, как компьютерная графика, обработка изображений, географические информационные системы и молекулярное моделирование. Современная реализация алгоритма была предложена Препаратой (Preparata) в 1975 г.

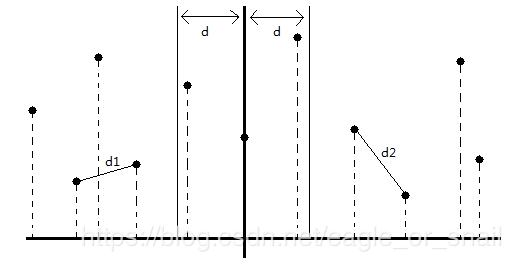
**Основной принцип устройства. Особенности.**

Формулировка задачи очень проста: для заданных n точек на плоскости найти пару точек, расположенных ближе друг к другу.

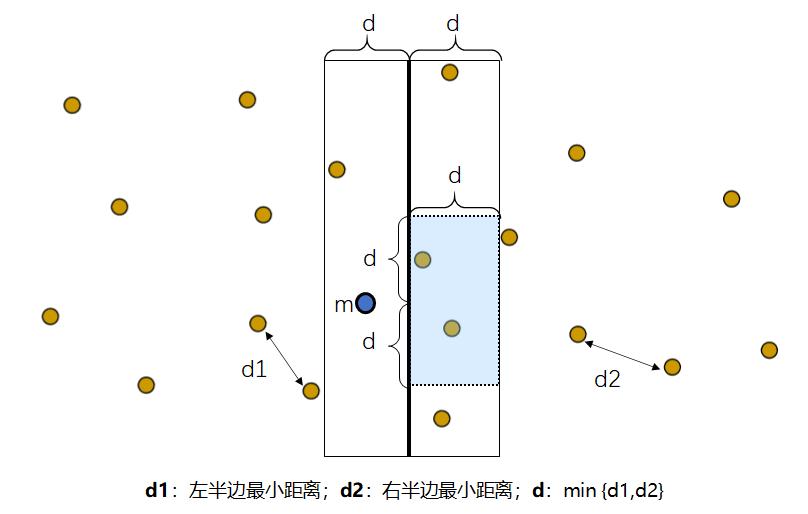
Очевидно, существует решение O(n2) — вычислить расстояния между каждой парой точек и выбрать минимум. Существует ли алгоритм, который был бы асимптотически более быстрым, чем квадратичный? Приведенный ниже алгоритм имеет сложность O(n log n)

Построим алгоритм по общей схеме алгоритмов "разделяй-и-властвуй": алгоритм оформляем в виде рекурсивной функции, которой передаётся множество точек; эта рекурсивная функция разбивает это множество пополам, вызывает себя рекурсивно от каждой половины, а затем выполняет какие-то операции по объединению ответов. Операция объединения заключается в обнаружении случаев, когда одна точка оптимального решения попала в одну половину, а другая точка — в другую (в этом случае рекурсивные вызовы от каждой из половинок отдельно обнаружить эту пару, конечно, не смогут)

Разбивать множество точек на два будем согласно их x-координатам: фактически мы проводим некоторую вертикальную прямую, разбивающую множество точек на два подмножества примерно одинаковых размеров.



Рекурсивно находим минимальное расстояние между точками справа и слева и берем наименьшее из них (пусть это будет расстояние d). Худший случай, когда точки с минимальным расстоянием находятся с разных сторон. В этом случае берем точки слева и справа от нашего разделения на расстоянии d. И сортируем уже по координате y. И в этом промежутке уже ищем минимальное расстояние.

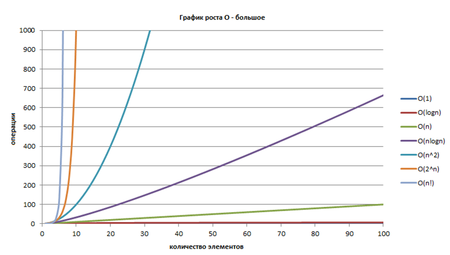


**Оценка временной сложности.**

1. Если рекурсивной функции передаётся множество из n точек, то стадия объединения должна работать не более, чем O(n), тогда асимптотика всего алгоритма T(n) будет находиться из уравнения:

 T(n) = 2 T(n/2) + O(n). 

Решением этого уравнения, как известно, является T(n) = O (n \log n).

1. Исходная сортировка P по х и y выполняется за время O(n log n). Время выполнения оставшейся части алгоритма удовлетворяет рекуррентному отношению, а следовательно, равно O(n log n).

**Выводы. Плюсы и минусы алгоритма, его применимость.**

Плюсы:

* «Разделяй и властвуй» — это мощный инструмент для решения концептуально сложных задач: все, что требуется для этого, — это найти случай разбивания задачи на подзадачи, решения тривиальных случаев и объединения подзадач в исходную задачу.
* Алгоритмы «Разделяй и властвуй» естественным образом стремятся эффективно использовать кэш память. Причина заключается в том, что как только подзадача достаточно мала, она и все ее подзадачи могут в принципе быть решены в кэше, не обращаясь к более медленной основной памяти.
* Базовый вариант имеет размер, ограниченный постоянной, то работа по разбиению задачи и объединению частичных решений пропорциональна размеру задачи n.

Минусы:

* Рекурсия. Алгоритмы «Разделяй и властвуй» естественным образом применяются в виде рекурсивных методов. В этом случае частные подзадачи, ведущие к той, которая в настоящее время решается, автоматически сохраняются в стеке вызовов процедур.
* Размер стека. В рекурсивных реализациях алгоритмов «Разделяй и властвуй» необходимо убедиться, что для стека рекурсии выделено достаточно памяти, иначе выполнение может завершиться неудачей из-за переполнения стека.

Применимость.

Для базовых вычислительных примитивов для геометрических задач. Этот алгоритм заложил основу области вычислительной геометрии и проникли в такие области, как компьютерная графика, обработка изображений, географические информационные системы и молекулярное моделирование

**Список использованной литературы:**

1. Кормен. Алгоритмы - построение и анализ.
2. Седжвик. Фундаментальные алгоритмы на C++
3. Препарата, Шеймос. Вычислительная геометрия: введение
4. <https://ru.wikipedia.org>

**Код программы:**

import java.util.\*;  
  
class Point {  
 int x;  
 int y;  
  
 public Point(int x, int y) {  
 this.x = x;  
 this.y = y;  
 }  
  
 public static double distance(Point ri, Point le) {  
 return Math.*sqrt*((ri.x - le.x)\*(ri.x - le.x) + (ri.y - le.y)\*(ri.y - le.y));  
 }  
 private static Object[] badDistance(Point[] list, int begin, int end) {  
 Object[] result = new Object[3];  
 result[0] = Double.*MAX\_VALUE*;  
 for(int i = begin; i <= end; i++) {  
 for(int j= i+1; j <= end; j++) {  
 // Находим расстояние каждой пары точек  
 double distance = Point.*distance*(list[i],list[j]);  
 if((double)result[0] >= distance) {  
 result[0] = distance;  
 result[1] = list[i];  
 result[2] = list[j];  
 }  
 }  
 }  
 return result;  
 }  
 public static Object[] seekDistance(Point[] list) {  
 // Сортируем точки в массиве по x  
 *quiSort*(list, 0, list.length - 1, "x");  
 // Рекурсивное решение  
 return *ClosestDistance*(list, 0, list.length - 1);  
 }  
  
 private static Object[] ClosestDistance(Point[] xArray, int begin, int end) {  
 Object[] result = null;  
  
 // Если количество точек меньше четырех, используем грубую сортировку(каждую дистанцию для каждой пары точек)  
 if(end - begin + 1 < 4) {  
 return *badDistance*(xArray, begin, end);  
 }  
 // Находим среднее значение  
 int midIndex = (begin+end)/2;  
 // Решаем левую половину  
 Object[] minLeft = *ClosestDistance*(xArray, begin, midIndex);  
 // Решаем правую половину  
 Object[] minRight = *ClosestDistance*(xArray, midIndex+1, end);  
 // Находим минимальное расстояние из левой и правой стороны  
 if((double)minLeft[0] < (double)minRight[0]) {  
 result = minLeft;  
 }else {  
 result = minRight;  
 }  
 // Перезаписываем кратчайшее расстояние  
 double minDistance = (double)result[0];  
  
 // Делим вертикальную полосу в соответствии с массивом минимальных значений  
 // Создаем два массива для хранения данных в области  
 // По принципу голубятни до 6 расчетов на каждую точку  
 ArrayList<Point> lList = new ArrayList<Point>();  
 ArrayList<Point> rList = new ArrayList<Point>();  
 int index = begin;  
 while(index <= end && xArray[index].x <= xArray[midIndex].x - minDistance) {  
 index++;  
 }  
 while(index <= end && xArray[index].x <= xArray[midIndex].x) {  
 // Точка в левой половине  
 lList.add(xArray[index]);  
 index++;  
 }  
 while( index <= end && xArray[index].x <= xArray[midIndex].x + minDistance) {  
 // Точка в правой половине  
 rList.add(xArray[index]);  
 index++;  
  
 }  
 // Преобразуем в массив  
 Point[] lArray = new Point[lList.size()];  
 Point[] rArray = new Point[rList.size()];  
 lList.toArray(lArray);  
 rList.toArray(rArray);  
 // Сортировать по y  
 *quiSort*(lArray, 0, lArray.length-1, "y");  
 *quiSort*(rArray, 0, rArray.length-1, "y");  
  
 // Рассчитываем расстояние до третьей пары точек  
 double minMid = minDistance ;  
 for(int i=0; i< lArray.length; i++){  
 for(int j=0; j< rArray.length; j++) {  
 // Когда значение y слишком мало, переходим к следующему циклу  
 if((lArray[i].y - rArray[j].y) < 0 - minDistance) {  
 continue;  
 }  
 // Когда расстояние между двумя точками больше минимального значения, выходим из этого цикла  
 if((lArray[i].y - rArray[j].y) > minDistance) {  
 break;  
 }  
 // Выбираем минимальное значение  
 minMid = Point.*distance*(lArray[i], rArray[j]);  
 if(minMid < minDistance) {  
 minDistance = minMid;  
 // Сохраняем пары расстояний и точек  
 result[0] = minDistance;  
 result[1] = lArray[i];  
 result[2] = rArray[j];  
 }  
 }  
 }  
 return result;  
 }  
 private static Point[] quiSort(Point[] list, int i, int j, String type) {  
 // Если длина больше единицы, использовать рекурсию для сортировки  
 if(i < j) {  
 int mid = *onceQuiSort*(list, i, j, type); // Находим элемент, по которому будем разделять  
 // Сортировка слева  
 *quiSort*(list, i, mid-1, type);  
 // Сортировка справа  
 *quiSort*(list, mid+1, j, type);  
 }  
 return list;  
 }  
  
 private static int onceQuiSort(Point[] list, int i, int j, String type) {  
 // Для первой сортировки по х  
 if(type.equals("x")) {  
 Point point = list[i];  
 while(i != j) {  
 while(i != j && list[j].x >= point.x) {  
 j--;  
 }  
 list[i] = list[j];  
 while(i != j && list[i].x <= point.x) {  
 i++;  
 }  
 list[j] = list[i];  
 }  
 list[i] = point;  
 }else {  
 // Вторая сортировка по y  
 Point point = list[i];  
 while(i != j) {  
 while(i != j && list[j].y >= point.y) {  
 j--;  
 }  
 list[i] = list[j];  
 while(i != j && list[i].y <= point.y) {  
 i++;  
 }  
 list[j] = list[i];  
 }  
 list[i] = point;  
 }  
 return i;  
 }  
 public static String toStringg(Point p){  
 return ("Point("+p.x+","+p.y+")");  
 }  
 public static void main(String[] args){  
 Point[] list = new Point[3];  
 list[0] = new Point(1,1);  
 list[1] = new Point(1,2);  
 list[2] = new Point(3,4);  
 Object[] result = *seekDistance*(list);  
 System.*out*.println ("Кратчайшее расстояние:" + result[0]);  
 System.*out*.println ("Соответствующая пара точек:" + *toStringg*((Point) result[1]) + "," + *toStringg*((Point) result[2]));  
 }  
}

**Ссылка на репозиторий** (все входные данные): https://github.com/kk0kc/lisa\_asd/tree/master/SecondSemestrovka