Университет ИТМО

Факультет ПИиКТ

Алгоритмы и структуры данных

Лабораторная работа №2

Выполнила: Наумова Надежда Александровна

Группа P3201

Преподаватель: Тропченко Андрей Александрович

Санкт-Петербург

2020 г.

## 1207. Медиана на плоскости

## Постановка задачи:

На плоскости находятся *N* точек (*N* чётно). Никакие три точки не лежат на одной прямой. Ваша задача — выбрать две точки так, что прямая линия, проходящая через них, делит множество точек на две части одинакового размера.

### Исходные данные

Первая строка содержит целое число *N* (4 ≤ *N* ≤ 10 000). Каждая из следующих *N* строк содержит пары целых чисел *xi*, *yi* (−106 ≤ *xi*, *yi* ≤ 106) — координаты *i*-й точки.

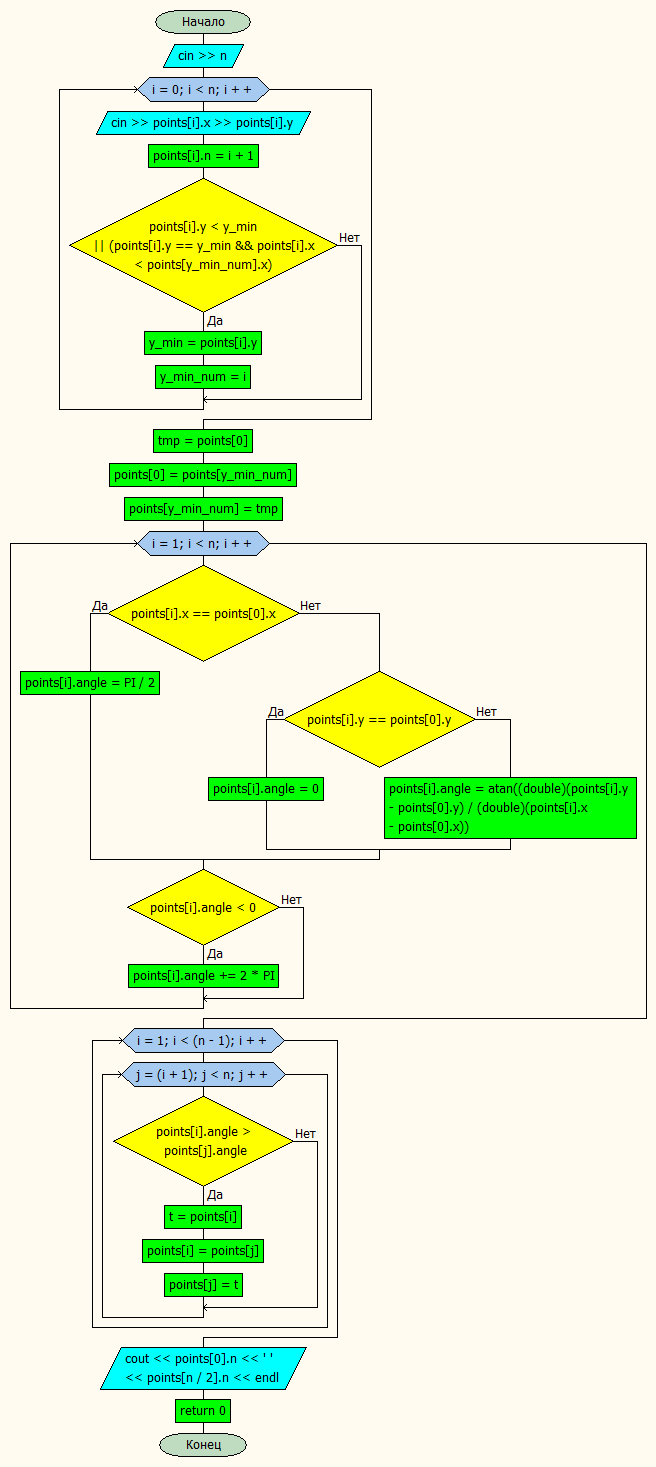
### Результат

Выведите номера выбранных точек.

## Описание решения

Для того, чтобы разделить на 2 части плоскость, найдем самую нижнюю точку, которая будет находиться как можно левее. Узнаем угол до каждой из оставшихся точек и отсортируем эти значения по величине угла. В таком случае мы сможем найти ту точку, через которую можно провести прямую, чтобы разделить плоскость так, чтобы половина оставшихся точек находились выше этой прямой, а вторая половина – ниже. Это возможно сделать из-за того, что никакие три точки не лежат на одной прямой.

## Схема решения



## Исходный код

// подключение необходимых библиотек  
#include <iostream>  
#include <cmath>  
  
// использование пространства имен std  
using namespace std;  
  
// структура, хранящая данные о точке: ее координаты, номер и угол  
// между осью OX и прямой, содержащей минимальную точку и текущую  
typedef struct {  
 long x;  
 long y;  
 long n;  
 double angle;  
} point;  
  
// объявление переменных для хранения ординаты и номера минимальной точки  
long y\_min = 1000001;  
long y\_min\_num = 0;  
// объявление константы для числа π  
const double PI = 3.14159265358979323846;  
  
int main() {  
  
 //объявление переменной для хранения кол-ва точек  
 int n;  
  
 //считывание кол-ва точек  
 cin >> n;  
  
 //объявление массива точек на n эл-тов  
 point points[n];  
  
 //считывание координат точки, присвоение ей номера  
 for (long i = 0; i < n; i ++) {  
 cin >> points[i].x >> points[i].y;  
 points[i].n = i + 1;  
  
 // поиск минимальной (нижней левой) точки (д.б. строго минимальной по y,  
 // по х - возможно и не самая левая), запись ее номера и ординаты

// в соответствующие переменные  
 if (points[i].y < y\_min ||  
 (points[i].y == y\_min && points[i].x < points[y\_min\_num].x)) {  
 y\_min = points[i].y;  
 y\_min\_num = i;  
 }  
 }  
  
 //для удобства поменяем местами первый элемент массива и минимальную точку  
 point tmp = points[0];  
 points[0] = points[y\_min\_num];  
 points[y\_min\_num] = tmp;  
  
 //вычисляем углы, приводим их к одному способу выражения -   
 // положительные (если отрицательный, добавим к нему полный оборот),   
 // отдельно рассмотрим случаи для углов 0 и 90  
 for (long i = 1; i < n; i ++) {  
 if (points[i].x == points[0].x)  
 points[i].angle = PI/2;  
 else if (points[i].y == points[0].y)  
 points[i].angle = 0;  
 else  
 points[i].angle = atan((double)(points[i].y - points[0].y) / (double)(points[i].x - points[0].x));  
  
 if (points[i].angle < 0) points[i].angle += 2 \* PI;  
 }  
  
 // сортировка пузырьком по углу  
 for (long i = 1; i < (n - 1); i ++) {  
 for (long j = (i + 1); j < n; j ++) {  
 if (points[i].angle > points[j].angle) {  
 point t = points[i];  
 points[i] = points[j];  
 points[j] = t;  
 }  
 }  
 }  
  
 // вывод результата  
 cout << points[0].n << ' ' << points[n/2].n << endl;  
  
 return 0;  
}

## Вывод:

в данной задаче я потренировалась в применении структуры, реализовала алгоритм сортировки пузырьком, а также освежила навыки решать геометрическую задачу.

## **1322. Шпион**

## Постановка проблемы

Спецслужбы обнаружили действующего иностранного агента. Шпиона то есть. Установили наблюдение и выяснили, что каждую неделю он через Интернет посылает кому-то странные нечитаемые тексты. Чтобы выяснить, к какой информации получил доступ шпион, требуется расшифровать информацию. Сотрудники спецслужб проникли в квартиру разведчика, изучили шифрующее устройство и выяснили принцип его работы.

На вход устройства подается строка текста S1 = s1s2...sN. Получив ее, устройство строит все циклические перестановки этой строки, то есть S2 = s2s3...sNs1, ..., SN = sNs1s2...sN-1. Затем множество строк S1, S2, ..., SN сортируется лексикографически по возрастанию. И в этом порядке строчки выписываются в столбец, одна под другой. Получается таблица размером N × N. В какой-то строке K этой таблицы находится исходное слово. Номер этой строки вместе с последним столбцом устройство и выдает на выход.

Например, если исходное слово S1 = abracadabra, то таблица имеет такой вид:

1. aabracadabr = S11
2. abraabracad = S8
3. abracadabra = S1
4. acadabraabr = S4
5. adabraabrac = S6
6. braabracada = S9
7. bracadabraa = S2
8. cadabraabra = S5
9. dabraabraca = S7
10. raabracadab = S10
11. racadabraab = S3

И результатом работы устройства является число 3 и строка rdarcaaaabb.

Это все, что известно про шифрующее устройство. А вот дешифрующего устройства не нашли. Но поскольку заведомо известно, что декодировать информацию можно (а иначе зачем же ее передавать?), Вам предложили помочь в борьбе с хищениями секретов и придумать алгоритм для дешифровки сообщений. А заодно и реализовать дешифратор.

### Исходные данные

В первой и второй строках находятся соответственно целое число и строка, возвращаемые шифратором. Длина строки и число не превосходят 100000. Строка содержит лишь следующие символы: a-z, A-Z, символ подчеркивания. Других символов в строке нет. Лексикографический порядок на множестве слов задается таким порядком символов:

ABCDEFGHIJKLMNOPQRSTUVWXYZ\_abcdefghijklmnopqrstuvwxyz

Символы здесь выписаны в порядке возрастания.

### Результат

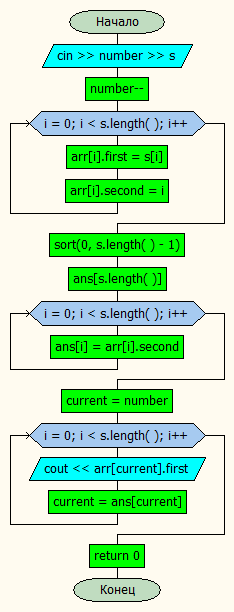
Выведите декодированное сообщение в единственной строке.

## Описание решения

Данная задача является реализацией обратного преобразования Барруоза-Виллера.

Для решения будем постепенно восстанавливать нашу исходную строку, прибавляя каждый раз по столбцу. Мы знаем последний столбец циклических сдвигов отсортированных, а также мы знаем первый столбец – его можно получить, отсортировав последний столбец. Циклически сдвинем полученные строки вправо и отсортируем полученные строки. После этого мы узнаем первые два символа отсортированных циклических сдвигов. Повторяя данный алгоритм k раз получим строку целиком. Заметим, что сортировка всегда будет одинаковой, а значит можно просто запомнить, куда какой элемент надо поставить и просто переставлять элементы. Заметим также, что нам нужна всего одна строка, а не все ее циклические сдвиги, поэтому будем строить только одну строку.

## Схема решения



## Исходный код

// подключение стандартной библиотеки  
#include <iostream>  
  
// использование пространства имен std  
using namespace std;  
  
// компаратор для сравнения либо в лексикографическом порядке,  
// либо, если буквы одинаковые, по номеру  
bool compare(pair<char, int> a, pair<char, int> b) {  
 if(a.first != b.first) {  
 return a.first < b.first;  
 }  
 return a.second < b.second;  
}  
  
// объявление массива на строку  
pair<char, int> arr[100001];  
  
// реализация алгоритма быстрой сортировки массива:  
// Пусть опорный элемент х - элемент массива, который стоит посередине между левой и правой границами  
// 1. Введем два указателя: i и j. В начале алгоритма они указывают, соответственно, на левый и правый конец последовательности.  
// 2. Будем двигать указатель i с шагом в 1 элемент по направлению к концу массива, пока не будет найден  
// элемент arr[i] >= x. Затем аналогичным образом начнем двигать указатель j от конца массива  
// к началу, пока не будет найден arr[j] <= x.  
// 3. Далее, если i <= j, меняем arr[i] и arr[j] местами и продолжаем двигать i,j по тем же правилам.  
// 4. Повторяем шаг 3, пока i <= j.  
// 5. Рекурсивно применить к подмассивам слева и справа от опорного элемента.  
void sort(int left, int right) {  
 int i = left;  
 int j = right;  
  
 pair<char, int> x = arr[(left + right) / 2];  
  
 while (i <= j) {  
 while (compare(arr[i], x)) {  
 i++;  
 }  
 while (compare(x, arr[j])) {  
 j--;  
 }  
  
 if (i <= j) {  
 swap(arr[i], arr[j]);  
 i++;  
 j--;  
 }  
 }  
  
 if (i < right) {  
 sort(i, right);  
 }  
 if (left < j) {  
 sort(left, j);  
 }  
}  
  
int main() {  
 // переменная для хранения строки, которую вернул шифратор  
 string s;  
   
 // переменная для хранения числа, которое вернул шифратор  
 int number;  
   
 // считывание числа и строки, которые вернул шифратор  
 cin >> number >> s;  
   
 // уменьшаем номер на 1 - для удобства нумерации с нуля  
 number--;  
  
 // записываем полученную строку в массив, который содержит пары: буква и

// ее номера в исходной строке  
 for (int i = 0; i < s.length(); i++) {  
 arr[i].first = s[i];  
 arr[i].second = i;  
 }  
  
 //сортируем массив arr  
 sort(0, s.length() - 1);  
  
 // заводим массив для хранения строки ответа  
 int ans[s.length()];  
  
 // запоминаем порядок, в котором переставляли  
 for (int i = 0; i < s.length(); i++) {  
 ans[i] = arr[i].second;  
 }  
  
 // переменная для хранения текущей позиции  
 int current = number;  
  
 // вывод результата по запомненным ранее перестановкам  
 for (int i = 0; i < s.length(); i++) {  
 cout << arr[current].first;  
 current = ans[current];  
 }  
  
 return 0;  
}

## Вывод:

в данной задаче я потренировалась в реализации алгоритма быстрой сортировки, а также поиске более эффективной реализации алгоритма

## **1726. Кто ходит в гости…**

## Постановка проблемы

Программный комитет школьных соревнований по программированию, проходящих в УрГУ — многочисленная, весёлая и дружная команда. Дружная настолько, что общения в университете им явно не хватает, поэтому они часто ходят друг к другу в гости. Все ребята в программном комитете очень спортивные и ходят только пешком.

Однажды хранитель традиций олимпиадного движения УрГУ подумал, что на пешие прогулки от дома к дому члены программного комитета тратят слишком много времени, которое могли бы вместо этого потратить на придумывание и подготовку задач. Чтобы доказать это, он решил посчитать, какое расстояние в среднем преодолевают члены комитета, когда ходят друг к другу в гости. Хранитель традиций достал карту Екатеринбурга, нашёл на ней дома всех членов программного комитета и выписал их координаты. Но координат оказалось так много, что хранитель не смог справиться с этой задачей самостоятельно и попросил вас помочь ему.

Город Екатеринбург представляет собой прямоугольник со сторонами, ориентированными по сторонам света. Все улицы города идут строго с запада на восток или с севера на юг, проходя через весь город от края до края. Дома всех членов программного комитета расположены строго на пересечении каких-то двух перпендикулярных улиц. Известно, что все члены комитета ходят только по улицам, поскольку идти по тротуару гораздо приятнее, чем по дворовым тропинкам. И, конечно, при переходе от дома к дому они всегда выбирают кратчайший путь. Программный комитет очень дружный, и все его члены ходят в гости ко всем одинаково часто.

### Исходные данные

Первая строка содержит целое число *n* — количество членов программного комитета (2 ≤ *n* ≤ 105). В *i*-й из следующих *n* строк через пробел записаны целые числа *xi*, *yi* — координаты дома *i*-го члена программного комитета (1 ≤ *xi*, *yi* ≤ 106).

### Результат

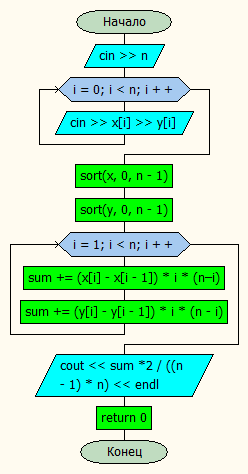
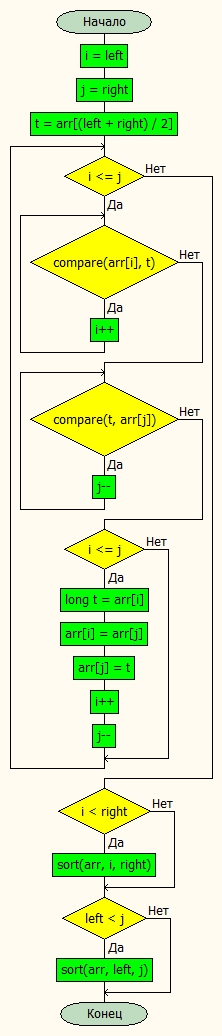
Выведите среднее расстояние, которое проходит член программного комитета от своего дома до дома своего товарища, округлённое вниз до целых.

## Описание решения

Заметим, что можно отдельно посчитать расстояние по x и по y, т. к. участники комитета ходят только по периметру клетки. Для нахождения найдём сумму расстояний между всеми домами и поделим её на количество” путей”.

Сохраним все координаты и отсортируем. Между каждой парой соседних домов узнаем расстояние r (по одной из координат) и узнаем, сколько человек по ней проходит (а именно i человек справа и (n - i) слева). Следовательно, суммарно по этой тропинке пройдут r \* i \* (n - i). Для поиска среднего расстояния этот весь путь следует разделить на суммарное количество проходов – оно вычисляется по формуле числа сочетаний из n по 2.

## Схема решения



## Исходный код

// подключение стандартной библиотеки  
#include <iostream>  
  
// использование пространства имен std  
using namespace std;  
  
// объявление переменной для кол-ва членов комитета, массивов с  
// координатами их домов, переменную хранения ответа  
long long n, x[100005], y[100005], sum = 0;  
  
// функция сравнения двух чисел  
bool compare(long long a1, long long a2) {  
 return a1 < a2;  
}  
  
// реализация алгоритма быстрой сортировки  
void sort(long long arr[], int left, int right) {  
 int i = left;  
 int j = right;  
  
 int t = arr[(left + right) / 2];  
  
 while (i <= j) {  
 while (compare(arr[i], t)) {  
 i++;  
 }  
 while (compare(t, arr[j])) {  
 j--;  
 }  
  
 if (i <= j) {  
 long long t = arr[i];  
 arr[i] = arr[j];  
 arr[j] = t;  
 i++;  
 j--;  
 }  
 }  
  
 if (i < right) {  
 sort(arr, i, right);  
 }  
 if (left < j) {  
 sort(arr, left, j);  
 }  
}  
  
  
int main() {  
 // считывание кол-ва участников комитета  
 cin >> n;  
  
 // для каждого участника - считываем координаты дома  
 for (int i = 0; i < n; i ++) {  
 cin >> x[i] >> y[i];  
 }  
  
 // сортировка массивов координат домов  
 sort(x, 0, n - 1);  
 sort(y, 0, n - 1);  
  
 // вычисление общего пройденного расстояния  
 for (int i = 1; i < n; i ++) {  
 sum += (x[i] - x[i - 1]) \* i \* (n – i);  
 sum += (y[i] - y[i - 1]) \* i \* (n - i);  
 }  
  
 // вывод ответа - среднего расстояния  
 cout << sum \* 2 /((n - 1) \* n) << endl;  
  
 return 0;  
}

## Вывод:

в данной задаче я потренировалась в реализации алгоритма быстрой сортировки, а также в применении этого алгоритма для более эффективного решения задачи.

## **1604. В Стране Дураков**

## Постановка проблемы

Главный бульдог-полицейский Страны Дураков решил ввести ограничение скоростного режима на автомобильной трассе, ведущей от Поля Чудес к пруду Черепахи Тортиллы. Для этого он заказал у Папы Карло *n* знаков ограничения скорости. Папа Карло слабо разбирался в дорожном движении и поэтому изготовил знаки с разными ограничениями на скорость: 49 км/ч, 34 км/ч, 42 км/ч, и т.д. Всего получилось *k* различных ограничений: *n*1 знаков с одним ограничением, *n*2 знаков со вторым ограничением, и т.д. (*n*1 + … + *nk* = *n*)

Бульдог-полицейский ничуть не расстроился, получив такие знаки, напротив, он решил извлечь из этого экономическую выгоду. Дело в том, что по Правилам дорожного движения Страны Дураков ограничение на скорость действует вплоть до следующего знака. Если на знаке написано число 60, это означает, что участок от данного знака до следующего нужно проехать ровно со скоростью 60 километров в час — не больше и не меньше. Бульдог распорядился расставить знаки так, чтобы обогатившимся на Поле Чудес автолюбителям во время своего движения по трассе приходилось как можно больше раз менять скорость. Для этого нужно расставить имеющиеся знаки в правильном порядке. Если Вы поможете бульдогу это сделать, то он готов будет поделиться с Вами частью своих доходов.

### Исходные данные

В первой строке дано число *k* — количество различных типов знаков с ограничением скорости (1 ≤ *k* ≤ 10000). Во второй строке через пробел перечислены целые положительные числа *n*1, …, *nk*. Сумма всех *ni* не превосходит 10000.

### Результат

Выведите *n* целых чисел в пределах от 1 до *k* — порядок, в котором нужно расставить по трассе имеющиеся знаки. Вне зависимости от того, какой знак стоит первым, считается, что, проезжая его, водитель меняет скорость, так как до этого ограничения не действовали. Если задача имеет несколько решений, выведите любое.

## Описание решения

На каждом шаге будем сортировать знаки по частоте появления и находить те, которые встречаются наиболее часто и те, которые встречаются реже всех. Затем будем ставить знаки таким образом, что за один шаг будем ставить наиболее и наименее часто встречающиеся знаки. Так будем делать, пока «указатели» (индексы) самого частого и самого редкого знака не встретятся, а когда это произойдет, выведем все оставшиеся знаки.

## Схема решения

## Исходный код

// подключение стандартной библиотеки  
#include <iostream>  
  
// использование пространства имен std  
using namespace std;  
  
// структура для знаков одного из видов, хранящая номер знака и их количество  
typedef struct {  
 int count;  
 int index;  
} sign;  
  
  
int main() {  
 // переменная для хранения кол-ва разных типов знаков

int n;  
   
 // считывание кол-ва разных типов знаков  
 cin >> n;  
  
 // объявление массива для хранения всех видов знаков  
 sign signs[n];  
   
 //переменные соответственно для хранения минимального и максимального   
 // кол-ва определенного вида знаков, их индексов   
 int min\_count = 10001, max\_count = 0, min\_index = -1, max\_index = -1;  
   
 for (int i = 0; i < n; i ++) {  
 //считываем информацию о каждом виде знаков  
 cin >> signs[i].count;  
 signs[i].index = i + 1;  
   
 //определяем вид знака, который встречается наиболее  
 // и наименее часто  
 if (signs[i].count >= max\_count) {  
 max\_count = signs[i].count;  
 max\_index = i;  
 }  
 if (signs[i].count < min\_count) {  
 min\_count = signs[i].count;  
 min\_index = i;  
 }  
 }  
  
 // пока, идя с разных концов нашего массива, указатели

// на начало и конец не встретятся  
 while (min\_index != max\_index && min\_count != 10001) {  
 // выводим знак, который встречается наиболее часто и знак, который

// встречается наиболее редко  
 printf("%i %i ", signs[max\_index].index, signs[min\_index].index);  
 // уменьшаем кол-во - мы их уже вывели на экран  
 signs[max\_index].count--;  
 signs[min\_index].count--;  
 // продолжаем поиски знаков, которые встречаются наиболее часто

// и наиболее редко  
 max\_count = 0;  
 min\_count = 10001;  
 for (int i = 0; i < n; i++) {  
 if (signs[i].count != 0 && signs[i].count >= max\_count) {  
 max\_count = signs[i].count;  
 max\_index = i;  
 }  
 if (signs[i].count != 0 && signs[i].count < min\_count) {  
 min\_count = signs[i].count;  
 min\_index = i;  
 }  
 }  
 }  
 // выводим те, что остались без пары  
 for (int i = 0; i < max\_count; i++) {  
 printf("%i ", signs[max\_index].index);  
 }  
 return 0;  
}

## Вывод:

в данной задаче я потренировалась в применении не сортировки как таковой, а нахождения максимального и минимального элементов в массиве к решению задачи.

## **1444. Накормить элефпотама**

## Постановка проблемы

Гарри Поттер сдаёт экзамен по предмету «Уход за магическими существами». Его задание — накормить карликового элефпотама. Гарри помнит, что элефпотамы отличаются прямолинейностью и невозмутимостью. Они настолько прямолинейны, что ходят строго по прямой, и настолько невозмутимы, что заставить их идти можно, только если привлечь его внимание к чему-нибудь действительно вкусному. И главное, наткнувшись на цепочку своих собственных следов, элефпотам впадает в ступор и отказывается идти куда-либо. По словам Хагрида, элефпотамы обычно возвращаются домой, идя в обратную сторону по своим собственным следам. Поэтому они никогда не пересекают их, иначе могут заблудиться. Увидев свои следы, элефпотам детально вспоминает все свои перемещения от выхода из дома (поэтому-то они и ходят только по прямой и лишний раз не меняют направление — так легче запоминать). По этой информации элефпотам вычисляет, в какой стороне расположена его нора, после чего поворачивается и идет прямо к ней. Эти вычисления занимают у элефпотама некоторое (довольно большое) время. А то, что некоторые невежды принимают за ступор, на самом деле есть проявление выдающихся вычислительных способностей этого чудесного, хотя и медленно соображающего животного!

Любимое лакомство элефпотамов — слоновьи тыквы, именно они и растут на лужайке, где Гарри должен сдавать экзамен. Перед началом испытания Хагрид притащит животное к одной из тыкв. Скормив элефпотаму очередную тыкву, Гарри может направить его в сторону любой оставшейся тыквы. Чтобы сдать экзамен, надо провести элефпотама по лужайке так, чтобы тот съел как можно больше тыкв до того, как наткнется на свои следы.

### Исходные данные

В первой строке входа находится число *N* (3 ≤ *N* ≤ 30000) — количество тыкв на лужайке. Тыквы пронумерованы от 1 до *N*, причем номер один присвоен той тыкве, у которой будет стоять элефпотам в начале экзамена. В следующих *N* строках даны координаты всех тыкв по порядку. Все координаты — целые числа от −1000 до 1000. Известно, что положения всех тыкв различны, и не существует прямой, проходящей сразу через все тыквы.

### Результат

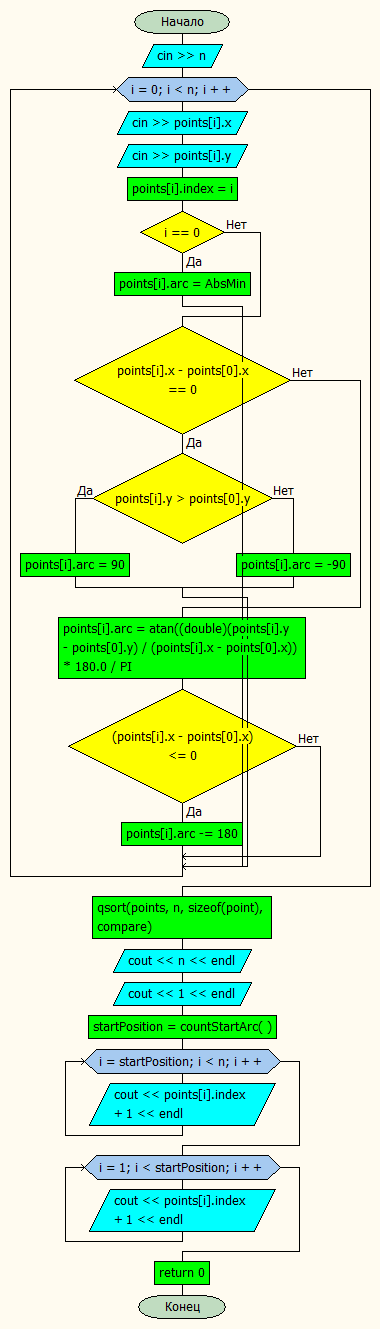
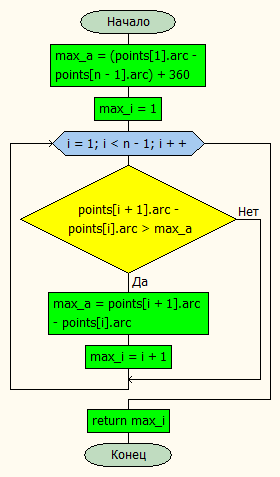
В первой строке выхода вы должны вывести *K* — максимальное количество тыкв, которое может съесть элефпотам. Далее по одному числу в строке выведите *K* чисел — номера тыкв в порядке их обхода. Первым в этой последовательности всегда должно быть число 1.

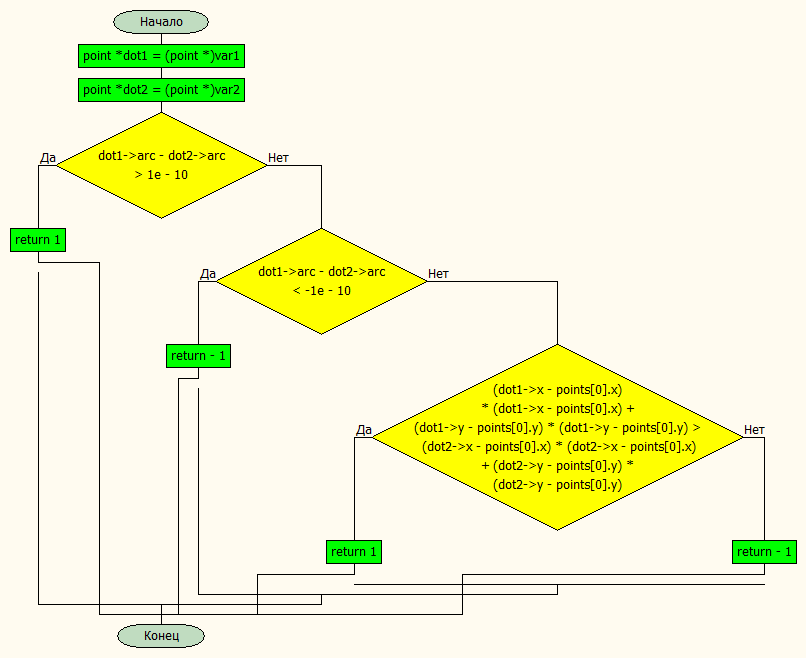
## Описание решения

Такие тыквы можно обойти всегда. Как это достигается? Найдем угол до каждой тыквы от начальной. Отсортируем углы. При одинаковом угле сначала элефпотам должен посетить ту, которая ближе к нему, потом по прямой дойти до остальных. Т. к. все тыквы отсортированы по величине угла от начальной, то пока элефпотам идет от текущей до следующей тыквы, он не пересечет свои следы (т. к. элефпотам еще ни разу не был ни в одной из точек плоскости, угол которой расположен между этими тыквами)

Важно проследить чтобы разность между соседними углами лучей из начальной точки всегда была меньше 180ти градусов. Такой угол может быть максимум один и если он есть, то надо начинать так, чтобы он не входил в наш путь.

## Схема решения





## Исходный код

//подключение необходимых библиотек  
#include <iostream>  
#include <cmath>  
  
// использование пространства имен std  
using namespace std;  
  
// объявление константы для числа π  
#define PI 3.141592653589793238462643383279502884197169399375105820974;  
#define AbsMin -9223372036854775807LL - 1;  
  
// структура, хранящая данные о точке: координаты, угол, номер  
struct point {  
 long int x;  
 long int y;  
 double arc;  
 int index;  
};  
  
// объявление переменной для хранения кол-ва точек и массива точек  
int n;  
point points[30001];  
  
//функция, поиска стартовой точки, с которой удобно будет пойти  
int countStartArc() {  
 // считаем, что максимальный угол - угол между первой и последней точками  
 double max\_a = (points[1].arc - points[n - 1].arc) + 360;  
 // переменная для хранения номера точки с максимальным углом  
 int max\_i = 1;  
 for (int i = 1; i < n - 1; i ++) {  
 // поиск максимального угла и номера  
 if (points[i + 1].arc - points[i].arc > max\_a) {  
 max\_a = points[i + 1].arc - points[i].arc;  
 max\_i = i + 1;  
 }  
 }  
 return max\_i;  
}  
  
//функция для сравнения точек по углу и, при равенстве углов, по расстоянию   
int compare(const void \*var1, const void \*var2) {  
 const point \*dot1 = (point \*) var1;  
 const point \*dot2 = (point \*) var2;  
 if (dot1->arc - dot2->arc > 1e-10) {  
 return 1;  
 } else if (dot1->arc - dot2->arc < -1e-10) {  
 return -1;  
 } else {  
 if ((dot1->x - points[0].x) \* (dot1->x - points[0].x) + (dot1->y - points[0].y) \* (dot1->y - points[0].y) >  
 (dot2->x - points[0].x) \* (dot2->x - points[0].x) + (dot2->y - points[0].y) \* (dot2->y - points[0].y)) {  
 return 1;  
 } else {  
 return -1;  
 }  
 }  
}  
  
int main() {  
 // считывание кол-ва точек  
 cin >> n;  
 for (int i = 0; i < n; i ++) {  
 //считывание координат точки, присвоение точке номера  
 cin >> points[i].x;  
 cin >> points[i].y;  
 points[i].index = i;  
  
 // первой точке присваивается минимальное значение угла  
 if (i == 0) {  
 points[i].arc = AbsMin;  
 continue;  
 }  
  
 // если первая точка и текущая имеют одинаковые абсциссы, то отдельно обозначим углы в 90 и -90 градусов  
 if (points[i].x - points[0].x == 0) {  
 if (points[i].y > points[0].y) {  
 points[i].arc = 90;  
 } else {  
 points[i].arc = -90;  
 }  
 continue;  
 }  
  
 // вычисляем значения остальных углов  
 points[i].arc = atan((double) (points[i].y - points[0].y) / (points[i].x - points[0].x)) \* 180.0 / PI;  
  
 //обработка случая, когда угол может лежать во II или III координатных

// четвертях  
 if ((points[i].x - points[0].x) <= 0) {  
 points[i].arc -= 180;  
 }  
 }  
  
 qsort(points, n, sizeof(point), compare);  
  
 // выводим кол-во точек, которые можно соединить  
 cout << n << endl;  
  
 // выводим стартовую точку  
 cout << 1 << endl;  
  
 // определяем ту точку, с которой удобнее делать обход  
 int startPosition = countStartArc();  
  
 // обход в конечном итоге замкнется, поэтому сначала выводим точки, которые  
 // окажутся в массиве после выбранной нами стартовой позиции  
 for (int i = startPosition; i < n; i ++) {  
 cout << points[i].index + 1 << endl;  
 }  
  
 // а затем те, которые оказались перед выбранной нами стартовой позицией  
 for (int i = 1; i < startPosition; i ++) {  
 cout << points[i].index + 1 << endl;  
 }  
 return 0;  
}

## Вывод:

в данной задаче я потренировалась в применении алгоритма сортировки к решению геометрической задаче.