Факультет программной инженерии и компьютерной техники

Алгоритмы и структуры данных

Сортировка

Преподаватели: Косяков Михаил Сергеевич, Тараканов Денис Сергеевич

Выполнил: Кульбако Артемий Юрьевич

P3212

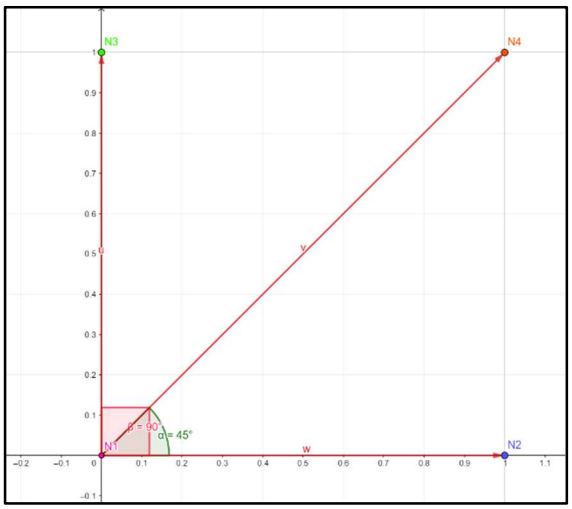
Санкт-Петербург
2020

1207. Медиана на плоскости

$$T(n) = O(n \log n)$$

```
#include <iostream>
#include <cmath>
#define PI 3.14159265358979323846264338327950288419716939937510582097494
using namespace std;

struct Point {
    int x;
    int y;
    int number;
    double angle;
};

void swap(Point* a, Point* b) {
    Point t = *a;
    *a = *b;
    *b = t;
}

void quickSort(Point array[], int left, int right) {
    int i = left, j = right;
    double pivot = array[(left + right) / 2].angle;
    while(i <= j) {
        while (array[i].angle < pivot) i++;
        while (array[j].angle > pivot) j--;
        if (i > j) break;
        swap(&array[i], &array[j]);
        i++;
        j--;
    }
    if (left < j) quickSort(array, left, j);
    if (i < right) quickSort(array, i, right);
}

int main() {
    int n;
    cin >> n;
    Point points[n];
    pair<int, int> minPoint = make_pair(INT_MAX, 0);
    for (int i = 0; i < n; i++) {
        cin >> points[i].x >> points[i].y;
        points[i].number = i + 1;
        if (points[i].y < minPoint.first || points[i].y == minPoint.first &&
points[i].x < points[minPoint.second].x) {
            minPoint.first = points[i].y;
            minPoint.second = i;
        }
    }
    swap(&points[0], &points[minPoint.second]);
```

```
    for (int 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) / (do
uble)(points[i].x - points[0].x));
        if (points[i].angle < 0) points[i].angle = points[i].angle + 2 * PI;
    }
    quickSort(points, 1, n - 1);
    cout << points[0].number << " " << points[n / 2].number << endl;
    return 0;
}
```



Создадим отдельную структуру `Point` для хранения сведений о точке. Чтобы прямая линия прошла через две точки и поделила множество точек на две части одинакового размера, нам для начала нужно выбрать опорную точку, которую примем за начало координат.

Возьмём минимальную по OX или OY точку $-$ N_1 и построим новые координатные оси

Построим вектора w, u, v до каждой из введённых пользователем точек (N2, N3, N4)

Рассчитываем углы относительно начала координат $-$ $\omega = 0°, \alpha = 45°, \beta = 90°$

Теперь нужно отсортировать массив точек по углу и выбрать из них среднюю. Прямая, проходящая через опорную и среднюю точки, поделила множество на две части.

1604. В стране дураков

$$\boldsymbol{T(n) = O(n)}$$

```
#include <iostream>
using namespace std;

struct SignBox {
    int amount;
    int index;
};

pair<int, int> findCurrentPair(SignBox array[], int k) {
    static pair<int, int> currentPair = make_pair(0, 0);
    int minAmount = INT_MAX, maxAmount = INT_MIN;
    for(int i = 0; i < k; i++) {
        if (array[i].amount > 0) {
            if (array[i].amount >= maxAmount) {
```

```
                maxAmount = array[i].amount;
                currentPair.second = i;
            }
            if (array[i].amount < minAmount) {
                minAmount = array[i].amount;
                currentPair.first = i;
            }
        }
    }
    return currentPair;
}

int main() {
    int k, minAmount = INT_MAX, maxAmount = INT_MIN, amountOfSigns = 0;
    cin >> k;
    SignBox signs[k];
    for (int i = 0; i < k; i++) {
        cin >> signs[i].amount;
        signs[i].index = i + 1;
        amountOfSigns += signs[i].amount;
    }
    pair<int, int> signsPair;
    while (amountOfSigns > 0) {
        signsPair = findCurrentPair(signs, k);
        if (signs[signsPair.second].amount > 0) {
            cout << signs[signsPair.second].index << " ";
            signs[signsPair.second].amount--;
            amountOfSigns--;
        }
        if (signs[signsPair.first].amount > 0) {
            cout << signs[signsPair.first].index << " ";
            signs[signsPair.first].amount--;
            amountOfSigns--;
        }
    }
    return 0;
}
```

По условию задачи, нам необходимо расставить дорожные знаки так, чтобы в полученной последовательности было как можно меньше знаков одного типа рядом. Тогда с большей вероятностью водитель не успеет переключить скорость и заплатит штраф Бульдогу-полицейскому. Создадим отдельную структуру для удобного хранения информации о знаках - `SignBox` - объект-обёртка, содержащая количество знаков определённого типа и их порядковый номер (можно использовать и объект `pair`, но это сделает код менее читаемым). Теперь попытается понять, как нужно расставлять знаки.

$$\text{Пусть } S = \sum a_i - \text{суммарное количество экземпляров всех знаков}$$

$$a_i - \text{количество экземпляров одного знака}$$

Тогда, если $a_i > \frac{S}{2} + 1$, то знаки где – то в последовательности всё же будут повторятся

Чтобы избежать этого в случайном месте результирующей последовательности, имеет смысл работать сначала с a_{max}, потом с a_{min}, в таком случае, повторения в последовательности могут быть только после того, как закончатся все $a_i < a_{max}$. Сделать это можно двумя способами:

A. Отсортировать массив типа `SignBox` по количеству экземпляров знака и брать элементы с концов этого массива.
B. Пока не кончились экземпляры знаков, находить знак с a_{min} и a_{max}, и ставить их в последовательность.

Я реализовал вариант B, потому что мне это показалось чуть-чуть проще, а полученный результат всё равно проходит по времени и памяти.

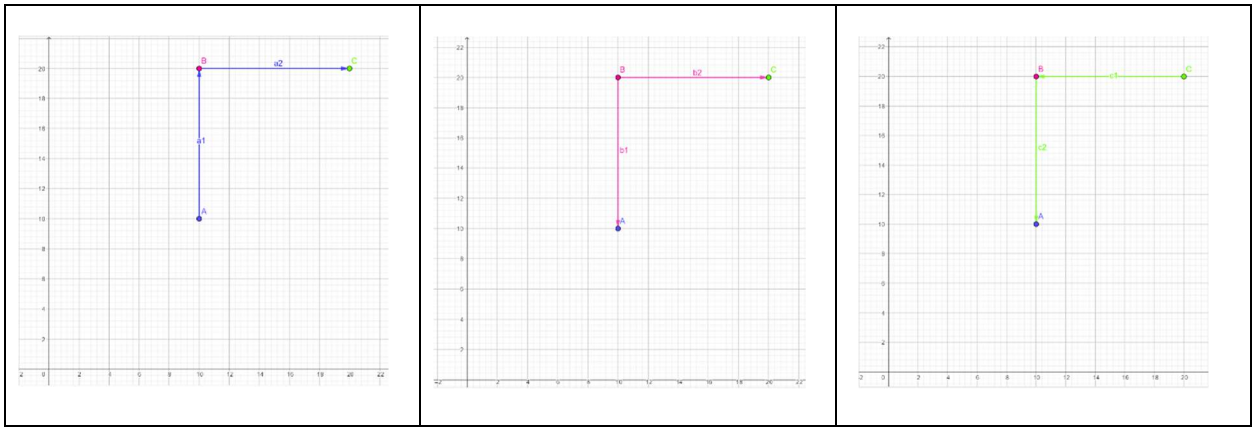
1726. Кто ходит в гости…

$$\boldsymbol{T(n) = O(n \log n)}$$

```
#include <iostream>
using namespace std;

void quickSort(int array[], int left, int right) {
    int i = left, j = right;
    int pivot = array[(left + right) / 2];
    while(i <= j) {
        while (array[i] < pivot) i++;
        while (array[j] > pivot) j--;
        if (i > j) break;
        int t = array[i];
        array[i] = array[j];
        array[j] = t;
        i++;
        j--;
    }
    if (left < j) quickSort(array, left, j);
    if (i < right) quickSort(array, i, right);
}

int main() {
    long long n, sum = 0;
    cin >> n;
    int xCoords[n];
    int yCoords[n];
    for (long long i = 0; i < n; i++) cin >> xCoords[i] >> yCoords[i];
    quickSort(xCoords, 0, n - 1);
    quickSort(yCoords, 0, n - 1);
    for (long long i = 1; i < n; i++) sum += ((xCoords[i] - xCoords[i-
1]) + (yCoords[i] - yCoords[i-1])) * (i) * (n - i) * 2;
    sum = sum / (n * (n - 1));
    cout << sum << endl;
    return 0;
```

```
}
```

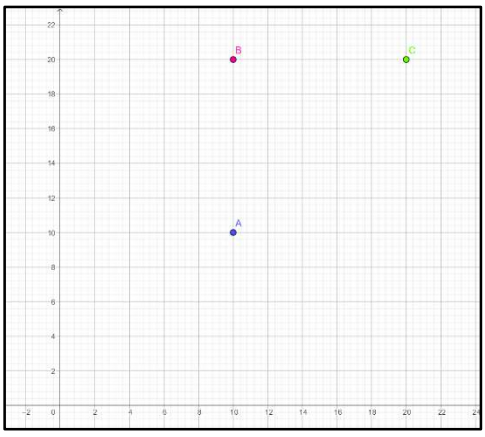
Построим карту происходящего, для лучшего понимания. Так как по условию задачи все члены программного комитета ходят в гости ко всем одинаково, то каждый дом нужно связать с каждым, исключительно прямыми линиями (можно перемещаться только параллельно OX и OY). Построим на ней вектора возможных перемещений между домами. Чтобы найти среднее растояние, которое проходит член программного комитета от своего дома до дома своего товарища, необходимо найти расстояние от его дома, до каждого из нужных домов, и повторить это с каждым членом комитета.



На этом этапе возникают две проблемы: 1-я - необходимость просчитывать заново длину дорожек, по которым уже кто-то ходил, и 2-я - как дать программе понять, что вектор между дорожками правильно направлен, чтобы не получать отрицательные значения длин векторов? Обе проблемы решаются сортировкой координат по возрастанию. Для этого реализуем алгоритм сортировки Хоара известный своим временем работы:

1. Выберем из массива опорный элемент - он будет равняться серединному элементу `(pivot = array[(left + right) / 2])`.
2. Разбиение: перераспределение элементов в массиве таким образом, что элементы меньше опорного помещаются перед ним, а больше или равные после.
3. Рекурсивно применить первые два шага к двум подмассивам слева и справа от опорного элемента. Рекурсия не применяется к массиву, в котором меньше двух элементов.

Решение 1-ой: Количество координат каждого типа равняется количеству домов на карте. Значит в отсортированном массиве для каждого элемента будет известна длина дорожки от `i` до `(n - i)` элемента (по которой уже ходил кто-то из членом клуба). Значит по каждой дороге пройдут (`длинаПути * i * (n - i)`) условных единиц расстояния.



Решение 2-ой: В отсортированном массиве мы всегда будем отнимать от большего x (или y) меньший (x или y), таким образом, не будем получать отрицательные значения (альтернативный вариант - функция `abs(x1 - x2)`).

Спроецируем карту города на оси OX и OY и посчитаем, суммарную длину всех путей. Чтобы найти средний путь члена комитета, поделим полученное число на количество существующих дорожек.