|  |
| --- |
| Университет итмо, кафедра вт |
| Лабораторная работа №2 по дисциплине «Алгоритмы и структуры данных» |
| «Сортировка» |
| Группа Р3202 |
| **Выполнила: Орлова Кристина Александровна** |
| **Преподаватель: Косяков Михаил Сергеевич** |

|  |
| --- |
| *31.05.18* |

**№1207**

Решение заключается в том, чтобы сперва найти самую левую точку (у которой координата x наименьшая), затем рассчитать тангенсы от этой точки до всех остальных, причем считаем, что, если координаты x обоих точек равны, а y2 > y (то есть рассматриваемая точка выше точки отсчета), то ее тангенс будет INT\_MAX, а если y2 < y (то есть рассматриваемая точка ниже точки отсчета), то ее тангенс будет INT\_MIN. Точке отсчета присваивается тангенс 0. Затем сортируем точки по тангенсам. Выбираем среди отсортированных точек ту, что лежит по середине (во время нахождения такой точки пропускаем точку отсчета). Затем выписываем эту и точку отсчета в ответ.   
При решении использовался массив структур point с полями int num, int x, int y, double tg.

|  |
| --- |
| #include <iostream> |
|  |  |
|  | using namespace std; |
|  |  |
|  | typedef struct point { |
|  | int num; |
|  | int x; |
|  | int y; |
|  | double tg; |
|  | } point; |
|  |  |
|  | int n; |
|  |  |
|  | int findLeft(const point \*array) { |
|  | point required = array[0]; |
|  |  |
|  | for (int i = 1; i < n; i++) { |
|  | if (array[i].x < required.x) { |
|  | required = array[i]; |
|  | } |
|  | } |
|  |  |
|  | return required.num; |
|  | } |
|  |  |
|  | void setTg(int index1, point \*array) { |
|  | array[index1].tg = 0; |
|  |  |
|  | for (int i = 0; i < n; i++) { |
|  | if (i != index1) { |
|  | if (array[i].x == array[index1].x) { |
|  | if (array[i].y > array[index1].y) { |
|  | array[i].tg = INT\_MAX; |
|  | } else { |
|  | array[i].tg = INT\_MIN; |
|  | } |
|  | } else { |
|  | array[i].tg = (double) (array[i].y - array[index1].y) / (array[i].x - array[index1].x); |
|  | } |
|  | } |
|  | } |
|  | } |
|  |  |
|  | void sort(point \*array) { |
|  | for (int i = 0; i < n - 1; i++) { |
|  | for (int j = i; j < n; j++) { |
|  | if (array[j].tg < array[i].tg) { |
|  | point timing = array[j]; |
|  |  |
|  | array[j] = array[i]; |
|  | array[i] = timing; |
|  | } |
|  | } |
|  | } |
|  | } |
|  |  |
|  | int findSecond(int index1, const point \*array) { |
|  | int half = n / 2; |
|  | int count = 0; |
|  | int index2 = 0; |
|  |  |
|  | for (int i = 0; i < n; i++) { |
|  | if (array[i].num != index1) { |
|  | count ++; |
|  | } |
|  |  |
|  | if (count == half) { |
|  | index2 = array[i].num; |
|  | break; |
|  | } |
|  | } |
|  |  |
|  | return index2; |
|  | } |
|  |  |
|  | void print(int index1, int index2) { |
|  | cout << index1 + 1 << " " << index2 + 1; |
|  | } |
|  |  |
|  | void getRequiredCoordinate(point \*array) { |
|  | int index1 = findLeft(array); |
|  | int index2; |
|  |  |
|  | setTg(index1, array); |
|  | sort(array); |
|  | index2 = findSecond(index1, array); |
|  |  |
|  | print(index1, index2); |
|  | } |
|  |  |
|  | int main() { |
|  | cin >> n; |
|  | point array[n]; |
|  |  |
|  | for (int i = 0; i < n; i++) { |
|  | cin >> array[i].x; |
|  | cin >> array[i].y; |
|  | array[i].num = i; |
|  | } |
|  |  |
|  | getRequiredCoordinate(array); |
|  | return 0; |
|  | } |

**№1604**

Массив количеств разных типов знаков сортируется по убыванию. Затем постепенно декриментируем элементы массива до тех пор, пока они все не обнулятся, по такому алгоритму:  
1) если максимальных кучек несколько, то чередуем знаки из них. (в этом случае мы постепенно обнулим все кучки, так как рано или поздно следующая станет входить в группу максимальных кучек).  
2) если максимальная кучка одна и следующая за ней максимальная кучка одна, то чередуем знаки из первой и второй кучки до тех пор, пока вторых максимальных кучек не станет несколько и перейдем к 3 пункту. (если она так и останется одна, то это означает, что у нас всего было 2 кучки по условию и сперва обнулится она, а затем придется дописывать оставшиеся знаки из первой кучки до ее обнуления).  
3) если максимальная кучка одна и следующих за ней максимальных кучек несколько, то чередуем знаки из первой максимальной и последней максимальной из второй группы кучек. (таким образом мы придем к тому, что первых максимальных кучек станет несколько и перейдем к пункту 1).  
4) после пункта 2 в одном из случаев можно перейти к этому пункту или же в том случае, если у нас по условию всего была 1 кучка.  
в этом пункте нужно выписать оставшиеся знаки без всякого чередования, так как остался всего 1 тип.

|  |
| --- |
| #include <iostream> |
|  |  |
|  | using namespace std; |
|  |  |
|  | typedef struct sign { |
|  | int count; |
|  | int number; |
|  | } sign; |
|  |  |
|  | typedef struct heap { |
|  | int countOfSigns; |
|  | int countOfTypes; |
|  | } heap; |
|  |  |
|  | int n; |
|  |  |
|  | // returns array sorted in descending order |
|  | void sort(sign \*array) { |
|  | for (int i = 0; i < n - 1; i++) { |
|  | for (int j = i + 1; j < n; j++) { |
|  | if (array[j].count > array[i].count) { |
|  | sign timing = array[i]; |
|  | array[i] = array[j]; |
|  | array[j] = timing; |
|  | } |
|  | } |
|  | } |
|  | } |
|  |  |
|  | void print(const string &sequence) { |
|  | cout << sequence; |
|  | } |
|  |  |
|  | void findAllForHeaps(heap \*heaps, int size, const sign \*array) { |
|  | int j = 0; |
|  | heaps[0].countOfSigns = array[0].count; |
|  | heaps[0].countOfTypes = 1; |
|  |  |
|  | for (int i = 1; i < size; i++) { |
|  | heaps[i].countOfTypes = 0; |
|  | heaps[i].countOfSigns = 0; |
|  | } |
|  |  |
|  | for (int i = 1; i < n; i++) { |
|  | if (array[i].count == heaps[j].countOfSigns) { |
|  | heaps[j].countOfTypes ++; |
|  | } else { |
|  | j++; |
|  |  |
|  | if (j == size) { |
|  | break; |
|  | } |
|  |  |
|  | heaps[j].countOfSigns = array[i].count; |
|  | heaps[j].countOfTypes = 1; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | void getRequiredSequence(sign \*array) { |
|  | heap heaps[2]; |
|  | string sequence; |
|  |  |
|  | while (array[0].count != 0) { |
|  | findAllForHeaps(heaps, 2, array); |
|  |  |
|  | if (array[1].count == 0 || n == 1) { |
|  | for (int i = 0; i < array[0].count; i++) { |
|  | sequence += to\_string(array[0].number) + " "; |
|  | } |
|  | break; |
|  | } |
|  |  |
|  | if (heaps[0].countOfTypes > 1) { |
|  | for (int i = 0; i < heaps[0].countOfTypes; i++) { |
|  | array[i].count --; |
|  | sequence += to\_string(array[i].number) + " "; |
|  | } |
|  | } else if (heaps[0].countOfTypes == 1 && heaps[1].countOfTypes == 1) { |
|  | array[0].count --; |
|  | array[1].count --; |
|  | sequence += to\_string(array[0].number) + " " + to\_string(array[1].number) + " "; |
|  | } else if (heaps[0].countOfTypes == 1 && heaps[1].countOfTypes > 1) { |
|  | array[0].count --; |
|  | array[heaps[1].countOfTypes].count --; |
|  | sequence += to\_string(array[0].number) + " " + to\_string(array[heaps[1].countOfTypes].number) + " "; |
|  | } |
|  | } |
|  |  |
|  | print(sequence); |
|  | } |
|  |  |
|  | int main() { |
|  | cin >> n; |
|  | sign array[n]; |
|  |  |
|  | for (int i = 0; i < n; i++) { |
|  | cin >> array[i].count; |
|  | array[i].number = i + 1; |
|  | } |
|  |  |
|  | sort(array); |
|  |  |
|  | getRequiredSequence(array); |
|  |  |
|  | return 0; |
|  | } |

**№1726**

Идея задачи заключается в том, что мы от одного любого дома можем пройти к любому другому через другие любые дома. Таким образом нужно найти лишь единичные пути и умножить их на (i + 1) \* (n - i - 1) для получения суммарного пройденного расстояния. Единичные пути ищутся как разности координат по x и координат по y между ближайшими домами. Полученное суммарное расстояние делится на n \* (n - 1)

|  |
| --- |
| #include <iostream> |
|  | #include <algorithm> |
|  | #include <vector> |
|  |  |
|  | using namespace std; |
|  |  |
|  | long long n; |
|  |  |
|  | void print(long long average) { |
|  | cout << average; |
|  | } |
|  |  |
|  | void getAverage(vector<int> x, vector<int> y) { |
|  | long long average = 0; |
|  |  |
|  | for (int i = 0; i < n - 1; i++) { |
|  | average += ((long long) (x[i + 1] - x[i] + y[i + 1] - y[i]) \* 2 \* (i + 1) \* (n - i - 1)); |
|  | } |
|  |  |
|  | average /= n \* (n - 1); |
|  |  |
|  | print(average); |
|  | } |
|  |  |
|  | int main() { |
|  | cin >> n; |
|  | vector<int> x; |
|  | vector<int> y; |
|  |  |
|  | for (int i = 0; i < n; i++) { |
|  | int xMean; |
|  | int yMean; |
|  |  |
|  | cin >> xMean >> yMean; |
|  |  |
|  | x.push\_back(xMean); |
|  | y.push\_back(yMean); |
|  | } |
|  |  |
|  | sort(x.begin(), x.end()); |
|  | sort(y.begin(), y.end()); |
|  |  |
|  | getAverage(x, y); |
|  |  |
|  | return 0; |
|  | } |