Университет ИТМО
Кафедра ВТ

Задачи 2 (1 - 5)
Алгоритмы и Структуры Данных

Выполнил: Федоров Сергей
Группа: P3212

Санкт-Петербург
2020 г.

- # Задача 1 - Медиана на плоскости - 1207

Так как кол-во точек всегда четно и ни одна тройка точек не лежит на одной прямой, то разделить плоскость на две половины (по кол-ву точек) всегда будет возможно. Для разделения выберем самую левую точку, затем найдем углы от этой точки до остальных (угол можно например считать от горизонтальной оси), отсортируем углы и проведем линию до медианной точки.

Код:

```
//
// Created by Sergey Fedorov on 14/02/2020.
//

#include <iostream>
#include <cmath>
#include <algorithm>

using namespace::std;

typedef struct {
    double angle;
    int index;
} rel_point;

bool rel_point_compar(rel_point a, rel_point b){
    return a.angle < b.angle;
}

int main(){
    int n;
    cin >> n;

    //                  x →       y →       index
    int left_dot[3] = {99999999, 99999999, 0};

    int dots[n][2];
    for (int i = 0; i < n; ++i) {
        cin >> dots[i][0] >> dots[i][1];

        if (left_dot[0] > dots[i][0]) {
            left_dot[0] = dots[i][0]; left_dot[1] = dots[i][1]; left_dot[2] = i;
        }
    }

    rel_point angles[n-1];
    int k = 0;

    for (int i = 0; i < n; ++i) if (i != left_dot[2]) {
        double x = dots[i][0] - left_dot[0];
        double y = dots[i][1] - left_dot[1];

        angles[k++] = {atan(y/x), i};
    }

    sort(angles, angles + n - 1, rel_point_compar);

    cout << left_dot[2]+1 << ' ' << angles[(n-1)/2].index+1 << endl;
}
```

- **Задача 2 - Шпион - 1322**

Прелюдия: я знал что школьный курс по сжатию данных, мне когда-нибудь пригодится.

По сути шифрования исходной строки происходит путем преобразования Берруоза-Виллера. Тогда для того чтобы расшифровать закодированное сообщение, нужно применить обратное преобразование Барроуза — Уилера. Суть обратного преобразования заключается в последовательной сортировки исходного сообщения с последующим приписыванием слева его же самого:

Обратное преобразование			
Вход			
BNN.AA.A			
Добавление 1	Сортировка 1	Добавление 2	Сортировка 2
B N N . A A . A	A A A B N N . .	BA NA NA .B AN AN .. A.	AN AN A. BA NA NA .B ..

Добавление 3	Сортировка 3	Добавление 4	Сортировка 4
BAN NAN NA. .BA ANA ANA ..B A..	ANA ANA A.. BAN NAN NA. .BA ..B	BANA NANA NA.. .BAN ANAN ANA. ..BA A..B	ANAN ANA. A..B BANA NANA NA.. .BAN ..BA
Добавление 5	**Сортировка 5**	**Добавление 6**	**Сортировка 6**
BANAN NANA. NA..B .BANA ANANA ANA.. ..BAN A..BA	ANANA ANA.. A..BA BANAN NANA. NA..B .BANA ..BAN	BANANA NANA.. NA..BA .BANAN ANANA. ANA..B ..BANA A..BAN	ANANA. ANA..B A..BAN BANANA NANA.. NA..BA .BANAN ..BANA

Добавление 7	Сортировка 7	Добавление 8	Сортировка 8
BANANA. NANA..B NA..BAN .BANANA ANANA.. ANA..BA ..BANAN A..BANA	ANANA.. ANA..BA A..BANA BANANA. NANA..B NA..BAN .BANANA ..BANAN	BANANA.. NANA..BA NA..BANA .BANANA. ANANA..B ANA..BAN ..BANANA A..BANAN	ANANA..B ANA..BAN A..BANAN BANANA.. NANA..BA NA..BANA .BANANA. ..BANANA
Результат			
.BANANA.			

Однако учитывая наш порядок N, это достаточно дорого каждый раз сортировать весь список, а так же хранить всю матрицу символов.
Существует оптимизированное обратное преобразование. Отсортируем сообщение один раз, запоминая их изначальные позиции. Затем используя известное число строки с правильным словом начнем с соответствующего символа, используя его индекс как ссылку к следующему символу, и так повторим N раз.

Код:

```
//
// Created by Sergey Fedorov on 14/03/2020.
//
#include <iostream>
#include <cmath>
#include <algorithm>

using namespace::std;

bool letter_comprar(pair<char, int> a, pair<char, int> b){
    if (a.first == b.first){
        return a.second < b.second;
    } else {
        return a.first < b.first;
    }
}

int main() {

    int k;
    cin >> k;

    string encoded;
    cin >> encoded;

    int n = encoded.length();
    pair<char, int> letter_2_position[n];
```

```
    for (int i = 0; i < n; ++i) {
        letter_2_position[i] = (pair<char, int>) {encoded[i], i};
    }

    sort(letter_2_position, letter_2_position + n, letter_comprar);

    int back_i = k - 1;
    for (int i = 0; i < n; ++i) {
        pair<char, int> next_letter = letter_2_position[back_i];
        cout << next_letter.first;
        back_i = next_letter.second;
    }
    cout << endl;
}
```

- ## Задача 3 - В стране дураков - 1604

Отсортируем массив дорожных знаков по их кол-ву. Каждую итерацию работаем с первыми двумя элементами массива, выводя по соответствующему знаку и уменьшая их кол-во на 1. Затем два раза вызываем шаг сортировки пузырьком (за один шаг сортировки пузырьком можно поднять один максимальный элемент наверх на один), для того чтобы если в какой-то момент наши максимальные элементы перестали быть таковыми они заменились на другие. Если элементы и так максимальные, то сортировка и не начнется. В случае если у головного элемента кол-во элементов не положительно, выходим из цикла.

Код:

```
//
// Created by Sergey Fedorov on 13/03/2020.
//

#include <iostream>
#include <algorithm>

using namespace::std;

typedef struct {
    int number, quantity;
} sign;

bool sign_compar(sign a, sign b){
    return a.quantity > b.quantity;
}

void bubble_sort_step(sign* signs, int length, pair<int, int> index_pair){
    if (index_pair.second < length) {
        while(signs[index_pair.first].quantity < signs[index_pair.second].quantity) {
            sign first = signs[index_pair.first];
            sign second = signs[index_pair.second];
            signs[index_pair.second] = first;
            signs[index_pair.first] = second;
            index_pair.first++; index_pair.second++;
            if (index_pair.second == length) break;
        }
    }
}
```

```
int main(){
    int k;
    cin >> k;

    int n = 0;
    sign signs[k];

    for (int i = 0; i < k; i++) {
        int quantity;
        cin >> quantity;
        n += quantity;
        signs[i] = (sign) {i + 1, quantity};
    }

    sort(signs, signs + k, sign_compar);

    while (n--) {
        std::cout << signs[0].number << " ";
        signs[0].quantity--;
        if (k > 1 && signs[1].quantity ≠ 0) {
            std::cout << signs[1].number << " ";
            signs[1].quantity--;
        }

        // Two iterations of bubble sort will be enough to rise max element by 2
positions.
        bubble_sort_step(signs, k, {1, 2});
        bubble_sort_step(signs, k, {0, 1});

        if (signs[0].quantity ⩽ 0) break;
    }
    std::cout << endl;
}
```

- **Задача 4 - Накормить элефапотама - 1444**

Задача схожа с 1207, однако теперь углы определяются относительно уже заранее установленной начальной точки (тыквы). Находим углы, сортируем по углам (в случае если углы одинаковые то сортируем по расстоянию от начально точки). Затем обходим против часовой стрелки (в направлении увеличения углов) каждую точку. Таким образом всегда возможно обойти все тыквы, так как всегда будет угловой сектор где наша животина еще не была, а следующая тыква именно там, тем самым она никогда не наткнется на свои следы.

Пы.Сы. Надо быть острожнее с ситуацией, когда две соседних тыквы находятся на угловом расстоянии больше 180 градусов (если будет то только один такой случай). В таком случае нужно начать обход с большей из этих двух тыкв и идя против часовой стрелке закончить меньшей тыквой из этих двух.

Код:

```
//
// Created by Sergey Fedorov on 16/04/2020.
//

#include <iostream>
#include <algorithm>
#include <cmath>

#define PI 3.14159265
```

```cpp
using namespace::std;

// Commented encountered problems

typedef struct {
    double angle;
    double distance; // Didn't use distance in sort
    int index;
} angle_distance;

bool comprar(angle_distance a, angle_distance b){
    if (abs(a.angle - b.angle) <= 1e-10){ // Initially compared by equality :(
        return a.distance < b.distance;
    } else {
        return a.angle < b.angle;
    }
}

//void print_angle(angle_distance a){
//    printf("Angle{%f, %f, %i}\n", a.angle, a.distance, a.index);
//}

int main() {

    int n;
    cin >> n;

     if (n > 0) {

         pair<int, int> pumpkins[n];
         for (int i = 0; i < n; ++i) {
             int x, y;
             cin >> x >> y;
             pumpkins[i] = (pair<int, int>) {x, y};
         }

         angle_distance angles[n - 1];

         for (int i = 1; i < n; ++i) {
             double x = pumpkins[i].first - pumpkins[0].first;
             double y = pumpkins[i].second - pumpkins[0].second;

             double cur_angle = atan2(y, x) * 180.0 / PI;
             double distance = sqrt(x * x + y * y);

             angles[i - 1] = (angle_distance) {cur_angle, distance, i + 1};
         }

         sort(angles, angles + n - 1, comprar);
//         for (angle_distance angle : angles) print_angle(angle);

         cout << n << endl;
         cout << '1' << endl;

        // Didn't consider having more than 180 degree difference between dots.
         int breakpoint = -1;
         for (int i = 1; i < n - 1; i++) {
             if (angles[i].angle - angles[i - 1].angle >= 180.0) {
                 breakpoint = i - 1;
```

```
                break;
            }
        }

        if (breakpoint == -1) {
            for (int i = 0; i < n - 1; i++) {
                cout << angles[i].index << endl;
            }
        } else {
            for (int i = breakpoint + 1; i < n - 1; i++) {
                cout << angles[i].index << endl;
            }
            for (int i = 0; i ⩽ breakpoint; i++) {
                cout << angles[i].index << endl;
            }
        }
    }
}
```

- **Задача 5 - Кто ходит в гости - 1726**

Заметим, что можно отдельно посчитать расстояние по x и по y. 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 (по одной из координат) и количество человек, которые по ней проходят. Следовательно, суммарно по конкретному пути пройдут `r (расстояние) * I (справа человек) * (n-I) (слева человек)` .

Пы. Сы. Надо следить за порядком операций и целочисленным типом, так как может произойти переполнение int'a и тогда ответ, очевидно, будет неверным.

Код:

```
//
// Created by Sergey Fedorov on 26/03/2020.
//

#include <iostream>
#include <algorithm>

using namespace::std;

bool simple_compar(long long a, long long b){
    return a < b;
}

int main(){
    int n;
    cin >> n;

    long long xs[n];
    long long ys[n];

    for (int i = 0; i < n; ++i) {
        cin >> xs[i] >> ys[i];
    }

    long long sum = 0;
```

```
    sort(xs, xs + n, simple_compar);
    sort(ys, ys + n, simple_compar);

    for (long long i = 1; i < n; ++i) sum += ((long long) ((xs[i] - xs[i - 1]) + (ys[i] -
ys[i - 1]))) *
                                             (long long) i * (long long) (n - i);


    sum = ((long long) 2) * sum / ((long long) n * (long long) (n-1)); // Need to
complete ops in this order,
                                                                       // because working
with integers.
                                                                       // Less rounding,
the better.

    cout << sum << endl;
}
```