Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №2

по «Алгоритмам и структурам данных»

Базовые задачи / **Timus**

Выполнил:

Студент группы P32121

Гиниятуллин Арслан Рафаилович

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2023

Задачи с Timus’a, handle = **vtalgo23_giniiatullin_335089**

Задача 1207: Медиана

```
struct point {
    int x, y, pos;
} points[10001];

inline bool comp(point a, point b) {
    if (atan2(a.y, a.x) - atan2(b.y, b.x) == 0)
        return a.x < b.x;
    return atan2(a.y, a.x) < atan2(b.y, b.x);
}

void solve() {
    int n;
    std::cin >> n;
    int mny = 2e9, mn_pos = -1;
    for (int i = 0; i < n; ++i) {
        std::cin >> points[i].x >> points[i].y;
        points[i].pos = i;
        if (points[i].y < mny) {
            mny = points[i].y;
            mn_pos = i;
        }
    }

    std::swap(points[0], points[mn_pos]);
    for (int i = 1; i < n; ++i) {
        points[i].x -= points[0].x;
        points[i].y -= points[0].y;
    }
    std::sort(points + 1, points + n, comp);
    std::cout << mn_pos + 1 << ' ' << points[n / 2].pos + 1 << '\n';
}
```

Описание решения:

Найдем точкой с минимальной ординатой и сделаем ее началом отсчета – перенесем все оставшиеся точки в данную СО. Заметим, что теперь нам необходимо найти такой радиус вектор, который делит первую и вторую четверти на две половинки с одинаковым кол-вом точек внутри. Это можно сделать, отсортировав радиус векторы по полярному углу к оси абсцисс. По условию, зная, что на одной прямой не более двух точек, – ответом будет точка – с индексом n / 2 в отсортированном массиве. Угол, образующийся вектором с концом в points [n / 2], будет как раз делить плоскость на две полуплоскости, с равным числом точек в каждой

Асимптотика: O(nlogn) – сортировка + линейные преобразования

Задача 1322: Шпион

```
void solve() {
    int n;
    std::string s, sorted;
    std::cin >> n >> s;
    --n;
    size_t len = s.size();
    std::unordered_map<char, std::vector<std::size_t>> mp;
    std::unordered_map<std::size_t, std::size_t> reverse_mp;
    sorted = s;
    std::sort(sorted.begin(), sorted.end());
    for (std::size_t i = 0; i < len; ++i) {
        mp[sorted[i]].push_back(i);
    }
    for (size_t i = len - 1; i != -1; --i) {
        reverse_mp[i] = mp[s[i]].back();
        mp[s[i]].pop_back();
    }
    char res[len];
    for (std::size_t i = 0, j = len - 1, k = n; i < len; ++i, --j) {
        res[j] = s[k];
        k = reverse_mp[k];
    }
    for (auto &i: res)
        std::cout << i;
}
```

Описание решения:

Заметим, что у нас уже есть два столбца получившейся отсортированной матрицы циклических сдвигов строки – первый (отсортированный данный) и последний (данный). Легко восстановить ответ за O(n^3logn), просто добавляя к столбцу в начало данный нам последний столбец и, сортируя результат.

Попробуем оптимизировать решение.

Хотим на каждом шаге для строки i знать ее позицию после добавления символа i. Для этого можем узнать позиции символов для данной нам строки в лексикографическом

порядке, т. е. позицию символа данной строки в отсортированной. Сделаем словарь, где ключ – позиция символа в первом столбце (отсортированный индекс), значение – позиция в последнем столбце (отсортированный индекс). Можем заметить, что теперь ответ легко восстановить, зная лексикографический порядок и порядок добавления букв в строку. Позиция i-й строки (ключ) после добавление элемента i-го элемента столбца в конец и сортировки, будет j (значение i). Потому что изначально наш суффикс отсортирован (по предположению индукции), переход – добавление в конец элемента i-го столбца, тк позиция этого символа в отсортированном массиве – j, то после сортировки (лексикографически), наш суффикс + i-й элемент будет на позиции j.

Асимптотика: ~O(nlogn) – сортировка за nlogn и get/set в мапу ~O(1).