Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

Отчет по блоку №2 (timus)

По Алгоритмам и структурам данных

Выполнил:

Ступин Тимур Русланович

Группа № P3108

Поток № 1.3

Преподаватель:

Тараканов Денис Сергеевич

Санкт-Петербург 2025

Содержание

Задача 1 Накормить элефпотама

Решение

Заметим, что благодаря гарантии того что все точки не лежат на одной прямой, всегда можно обойти все вершины, достаточно только выбрать правильный порядок обхода. Данным порядком является обход по или против часовой стрелки. Чтобы получить такой порядок, нужно отсортировать точки по полярному углу и по расстоянию до начала координат (в случае равенства углов). При этом за начало координат удобно принять первую точку, так как обход в любом случае начнется с неё. Для переноса начала координат нужно просто покоординатно вычесть первую точку из всех остальных. Единственное что осталось решить, какая точка будет второй. Этот выбор будет определяющим, так как дальнейший обход определится однозначно (точки образуют цикл). Рассмотрим все соседние пары точек в отсортированном массиве (начальную не учитываем) и найдем точки, угол между которыми максимальный. Начнем обход со второй из этих точек, дойдем до наибольшей, после чего продолжим обход с наименьшей вплоть до ранее выбранной точки.

Сложность алгоритма

- По времени: Так как выполняется сортировка точек, сложность $O(NlogN)$
- По памяти: В памяти хранятся только точки. $O(N)$

Итого:

- По времени: $O(NlogN)$
- По памяти: $O(N)$

Код

```cpp
#include <bits/stdc++.h>
using namespace std;

struct point {
  int x;
  int y;
  int i;
};

constexpr double EPS = 1e-9;
constexpr int INF = 1e9;
constexpr double PI = 3.14159265358979323846;

bool comp(const point p1, const point p2) {
  const double a1 = atan2(p1.y, p1.x);
  const double a2 = atan2(p2.y, p2.x);
  if (abs(a1 - a2) < EPS) {
    return p1.x * p1.x + p1.y * p1.y < p2.x * p2.x + p2.y * p2.y;
  }
  return a1 - a2 < EPS;
}

int main() {
  int n;
  cin >> n;
  point first_point{};
  cin >> first_point.x >> first_point.y;
  vector<point> v(n - 1);
  for (int i = 0; i < n - 1; i++) {
    cin >> v[i].x >> v[i].y;
    v[i].i = i + 2;
  }
  for (int i = 0; i < n - 1; i++) {
    v[i].x -= first_point.x;
    v[i].y -= first_point.y;
  }
  sort(v.begin(), v.end(), comp);
  int k = -1;
  double max_ang = -2 * PI;
  for (int i = 0; i < n - 1; i++) {
    const int prev_i = (i - 1) < 0 ? (i - 1) + (n - 1) : (i - 1);
    double delta = abs(atan2(v[i].y, v[i].x) -
 atan2(v[prev_i].y, v[prev_i].x));
    if (delta - PI > EPS)
      delta = 2 * PI - delta;
    if (delta - max_ang > EPS) {
      k = i;
      max_ang = delta;
    }
```

```
50    }
51    cout << n << endl;
52    cout << 1 << endl;
53    for (int i = k; i < n - 1; i++) {
54      cout << v[i].i << endl;
55    }
56    for (int i = 0; i < k; i++) {
57      cout << v[i].i << endl;
58    }
59    return 0;
60  }
```

Задача 2 В Стране Дураков

Решение

Применим жадный подход. На каждом шаге выгоднее всего выбрать знак с наименьшим числом экземпляров и знак с наибольшим числом экземпляров. Для этого нужно уметь достаточно быстро получать минимум и максимум массива, а также поддерживать удаление и добавление элементов. Для этого отлично подходит `set`. Пока в нём больше одного элемента производим извлечения максимума и минимума, вывод, уменьшение на 1 и возвращение. Когда цикл завершен, если множество не пусто выводим всё что осталось подряд, эти знаки не с кем чередовать.

Сложность алгоритма

- По времени: Основной цикл проходит по всем знакам, но по условию их не больше `k`. Однако на каждой итерации выполняются действия с `set`, которые имеют сложность $O(logN)$. Итоговая сложность $O(NlogN)$
- Для хранения множества необходимо $O(N)$ памяти

Итого:

- По времени: $O(NlogN)$
- По памяти: $O(N)$

Код

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
  int k;
  cin >> k;
  set<pair<int, int>> s;
  for (int i = 0; i < k; i++) {
    int x;
    cin >> x;
    s.insert({x, i + 1});
  }
  while (s.size() > 1) {
    const auto it_x = s.begin();
    const auto it_y = prev(s.end());
    auto x = *it_x;
    auto y = *it_y;
    cout << y.second << " " << x.second << " ";
    x.first--;
    y.first--;
    s.erase(it_x);
    s.erase(it_y);
    if (x.first > 0)
      s.insert(x);
    if (y.first > 0)
      s.insert(y);
  }
  if (!s.empty()) {
    for (int i = 0; i < s.begin()->first; i++) {
      cout << s.begin()->second << " ";
    }
  }
  cout << endl;
  return 0;
}
```