Лабораторная работа № 8 по курсу дискретного анализа: Жадные алгоритмы

Выполнил студент группы М8О-308Б-21 МАИ *Белоносов Кирилл.*

Условие

1. Общая постановка задачи: Заданы длины N отрезков, необходимо выбрать три таких отрезка, которые образовывали бы треугольник с максимальной площадью.

2. Вариант задания(2):

   Формат ввода: На первой строке находится число N, за которым следует N строк с целыми числами-длинами отрезков.
   Формат вывода:Если никакого треугольника из заданных отрезков составить нельзя – 0, в противном случае на первой строке площадь треугольника с тремя знаками после запятой, на второй строке – длины трёх отрезков, составляющих этот треугольник. Длины должны быть отсортированы.

Метод решения

Для решения данной задачи, вспомним то свойство, что любая сторона треугольника должна быть меньше, чем сумма двух других сторон. Воспользуемся этим, отсортируем вектор сторон по убыванию. Таким образом, для любого элемента выполняется: $i \geq i+1$. Тогда выберем первый кортеж из 3 чисел и проверим их на существование треуголника, возникает 2 ситуации:

1. Треугольник не существует. Нам не имеет смысла проверять все остальные пары сторон для первого элемента кортежа, т.к. сумма других пар будет меньше либо равно сумме текущих пар, которые являются уже максимально возможной суммой. Значит мы можем сдвинуть наш "курсор"на одну позицию вправо.

2. Треугольник существует. Но он не обязательно имеет максимальную площадь, возможно в дальнейшем будут стороны, которые дадут большую площадь. Поэтому если текущая площадь больше сохранненой, то обновляем сохраненную

Описание программы

Считаем количество сторон. Считаем сами стороны, отсортировав их в порядке убывания. Если количество сторон меньше 3, то мы не сможем составить ответ и выведем 0. Иначе будем бежать по массиву и брать 3 соседних элемента и проверять их на существование треугольника

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

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.precision(3);
    int N;
    cin >> N;

    vector<double> a(N);
    for(double& i: a)
        cin >> i;

    sort(a.rbegin(), a.rend());

    double ans = 0;
    int x, y, z;
    if(N >= 3) {
        for(int i = 0; i < N - 2; ++i) {
            if(
                a[i] < a[i + 1] + a[i + 2] &&
                a[i + 1] < a[i] + a[i + 2] &&
                a[i + 2] < a[i] + a[i + 1]
            ) {
                double p = (a[i] + a[i+1] + a[i+2]) / 2;
                double current_square = sqrt(p * (p - a[i]) * (p -
                   a[i + 1]) * (p - a[i + 2]));
                if(current_square > ans) {
                    x = a[i + 2];
                    y = a[i + 1];
                    z = a[i];
                    ans = current_square;
                }
            }
        }
    }
    if(ans != 0) {
        cout << fixed << ans << "\n";
        cout << x << " " << y << " " << z;
    } else {
        cout << ans << "\n";
    }
    return 0;
}
```

Дневник отладки

Попытка 1

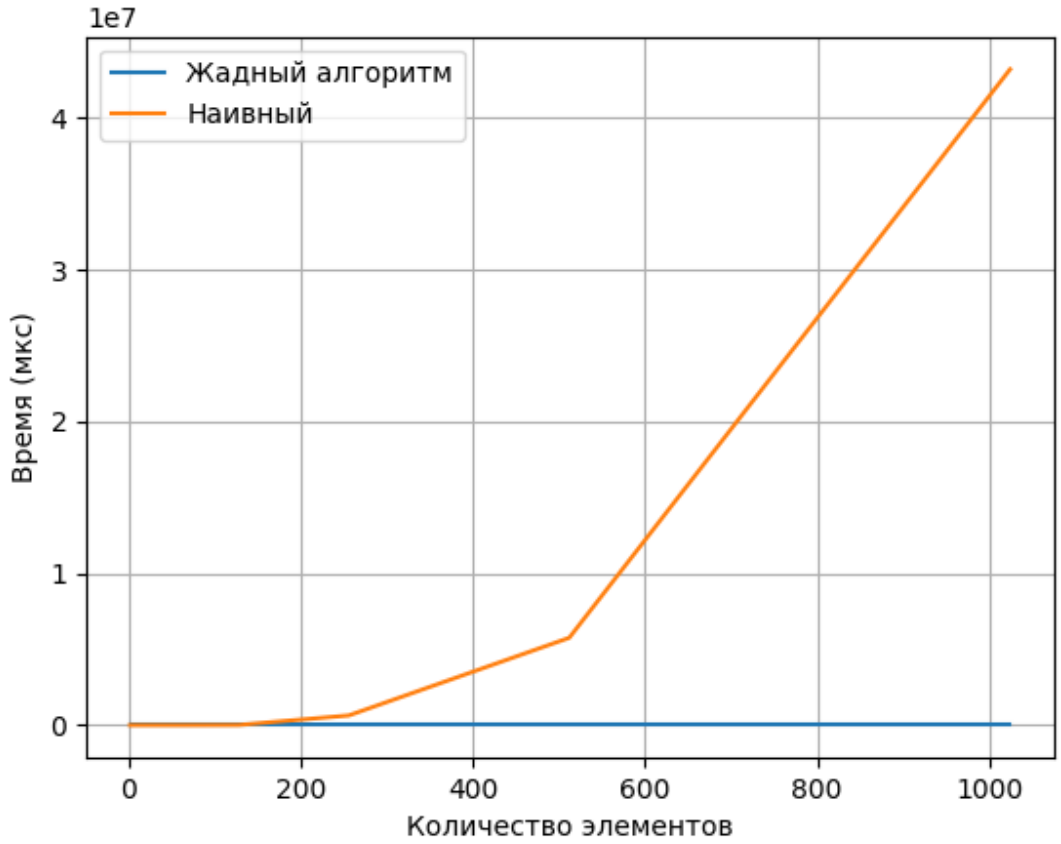
Программа зашла с первой попытки. Могла зайти со второй, но я вовремя увидел условие про 3 знака после запятой.

0.1 Результаты тестирования

Тест производительности

Тестирование программы производилось на 10 сгенерированных тестах Для сравнения был выбран наивный алгоритм:

Вставка:



Получившиеся результаты удивляют особо не удивляют, алгоритм за линию справился намного лучше, чем алгоритм за куб

Выводы

В данной лабораторной работе я разработал программу решения задачи, используя жадный алгоритм. Данное задание потребовало знания полученные в ходе изучения курса дискретного анализа и немного математики. Можно увидеть, что казалось бы сложные задачи имеют быстрое решение, опирающееся на математический аппарат или небольшие хитрости.

Как по мне, самой сложной частью решения такого типа задач как Жадные алгоритмы и динамическое программирование как раз и является нахождение таких хитрых зависимостей.