# Московский авиационный институт (национальный исследовательский университет)

## Факультет информационных технологий и прикладной математики

## Кафедра вычислительной математики и программирования


**Лабораторная работа №8 по курсу «Дискретный анализ»**


Студент: К. А. Калугин
Преподаватель: А. А. Кухтичев
Группа: М8О-307Б

Дата:
Оценка:
Подпись:


**Москва, 2021**

# Лабораторная работа №81

**Задача:** Разработать жадный алгоритм решения задачи, определяемой своим вариантом. Доказать его корректность, оценить скорость и объём затрачиваемой оперативной памяти.

Реализовать программу на языке C или C++, соответсвующую построенному алгоритму. Формат входных и выходных данных описан в варианте задания.

Заданы длины N отрезков, необходимо выбрать три таких отрезка, которые образовывали бы треугольник с максимальной площадью.

**Формат входных данных:** На первой строке находится число N, за которым следует N строк с целыми числами-длинами отрезков.

**Формат результата:** 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.

# 1 Описание

Основная идея жадных алгоритмов заключается в выборе самого оптимального решения на каждом шаге в надежде на оптимальное решение. У такого подхода есть и минусы, тк существуют задачи (например задача рюкзака), которые жадный алгорит решает неправильно.

В моем случае, необходимо найти такие стороны, чтобы площадь треугольника, который они образовывают была максимальной. Для этого я, на основании формулы Герона выяснил, что площадь тем больше, чем больше каждая из сторон треугольника. Таким образом, необходимо на каждом шаге выбирать самую большие возможные стороны из которых можно собрать треугольник.

# 2 Исходный код

Считаем длины сторон и запишем их в вектор. Отсортируем его. Сложность стандартной функции vector.sort равна O($nlog(n)$). После этого пройдем по вектору «рамкой» в три элемента, выбирая сначала наибольшиен стороны. Для каждой рамки проверим - могут ли стороны образовать треугольник и, если могут, посчитаем его площадь. Таким образом найдем максимальную площадь и стороны, которые ее дают. Сложность прохода по вектору - O($n - 2$). Таким образом, общая сложность алгоритма - O($n^2 * log(n)$).

Для хранения длин мы создаем вектор длинны n, так что используемый объем оперативной памяти не превышает $int * n$ байта.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <math.h>

using namespace std;

int main () {
    int n;
    cin >> n;
    long long int a = 0, b = 0, c = 0;
    long long int bestA = 0, bestB = 0, bestC = 0;
    double bestS = -1;
    vector <long long int> lines (n);
    for (int i = 0; i < n; i ++) {
        cin >> lines [i];
    }
    if (n >= 3) {
        sort (lines.begin (), lines.end ());
        for (int i = n - 1; i >= 2; i --) {
            if (lines [i] < lines [i - 1] + lines [i - 2]) {
                a = lines [i];
                b = lines [i - 1];
                c = lines [i - 2];
                //break;
            }
            if ((a != 0) && (b != 0) && (c != 0) && (a < b + c)) {//
                double p = double ((a + b + c)) / 2;
                double s = double (sqrt (p * (p - a) * (p - b) * (p - c)));
                if (s > bestS) {
                    bestS = s;
                    bestA = a;
                    bestB = b;
                    bestC = c;
                }
            }//
```

```
37          }
38      }
39      else {
40          cout << 0 << endl;
41      }
42      if (bestS > 0) {//
43          printf ("%.3lf\n", bestS);
44          cout << bestC << " " << bestB << " " << bestA << endl;
45      }
46      else {
47          cout << 0 << endl;
48      }//
49
50      return 0;
51  }
```

# 3 Консоль

```
PS C:\VSC\DA>.\a.exe
4
1
2
5
0
PS C:\VSC\DA>.\a.exe
3
3
4
5
6.000
3 4 5
```

# 4 Тест производительности

Наивный алгоритм перебирает все комбинации сторон, поэтому его сложность - O($n^3$). Сравним его эффективность и эффективность жадного алгоритма на тестах размера 4, 50 и 100.

```
PS C:\VSC\DA>g++ Lab8.cpp -pedantic -Wall -std=c++11 -Werror -Wno-sign-compare
-O2 -lm main.cpp -o solution
g++ B8.cpp -pedantic -Wall -std=c++11 -Werror -Wno-sign-compare -O2 -lm benchmark.cpp
-o benchmark
PS C:\VSC\DA>./solution <test1.txt
It took me 81 clicks (0.000081 seconds).
PS C:\VSC\DA>./benchmark <test1.txt
It took me 14 clicks (0.000014 seconds).
PS C:\VSC\DA>./solution <test2.txt
It took me 150 clicks (0.000150 seconds).
PS C:\VSC\DA>./benchmark <test2.txt
It took me 324 clicks (0.000324 seconds).
PS C:\VSC\DA>./solution <test3.txt
It took me 103 clicks (0.000103 seconds).
PS C:\VSC\DA>./benchmark <test3.txt
It took me 2721 clicks (0.002721 seconds).
```

Как видно, на малых значениях наивный алгоритм выигрывает, за счет отстутствия в нем сортировки, однако на больших тестах начинает стремительно проигрывать.

# 5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я научился следующим вещам:

Во-первых, иногда попытки излишней оптимизации вредны, тк могут повлиять на правильность работы программы.

Во-вторых, иногда стоит встраивать в программу предохранительные элементы, даже если согласно теоретическим рассуждениям, они не будут использоваться.

В-третьих, судя по всему, может существовать пара треугольников, в которой стороны или сторона одного меньше, чем у второго, при этом площадь — больше.

# Список литературы

[1] *Жадные алгоритмы.*
URL: `https://ru.wikipedia.org/wiki/Жадный`( : 23.11.2021).