

Лабораторная работа № 8 по курсу дискретного анализа: жадные алгоритмы

Выполнил студент группы М8О-308Б-22 Н. А. Ширяев

Условие

Кратко описывается задача:

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

2. 3

Метод решения

1. Считываем данные и кладем все длины отрезков в вектор.
2. Сортируем вектор, чтобы все длины были расположены в порядке возрастания.
3. Фиксируем две длины стороны и ищем длину для третьей стороны, которая максимизирует площадь.
4. Если на каком-то шаге находим, что площадь нового треугольника увеличилась, то обновляем значение площади, площадь считается по формуле Герона.

Описание программы

В моей программе один файл main.cpp:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <iomanip>
#include "cmath"

double area(int a, int b, int c) {
    double p = double (a + b + c) / 2;
    return sqrt(p * (p - a) * (p - b) * (p - c));
}

int main() {
    int N;
    std::cin >>
    N;

    if(N < 3) {
```

```
std::cout << 0;
```

```

        return 0;
    }

    std::vector<int> lengths;
    for(int i = 0; i < N; ++i) {
        int temp = 0;
        std::cin >> temp;
        lengths.push_back(temp);
    }
    std::sort(lengths.begin(),
lengths.end()); double maxTriangleArea =
0;
int A = 0, B = 0, C = 0;

    for(int i = 0; i < lengths.size() - 2; ++i) {
        for(int j = i + 1; j < lengths.size() - 1; ++j) {
            int a = lengths[i]; int
            b = lengths[j]; int c =
            lengths[j + 1];
            if (a < b + c && b < a + c && c < a + b)
                { if (area(a, b, c) > maxTriangleArea)
                    {
                        maxTriangleArea = area(a, b, c);
                        A = a;
                        B = b;
                        C = c;
                    }
                }
        }
    }

    std::vector<int> sortedLengths;
    sortedLengths.push_back(A);
    sortedLengths.push_back(B);
    sortedLengths.push_back(C);
    std::sort(sortedLengths.begin(), sortedLengths.end());
    if(A != 0){
        std::cout << std::fixed << std::setprecision(3) << maxTriangleArea
<< std::endl;
        std::cout << sortedLengths[0] << " " << sortedLengths[1] << " " <<
sortedLengths[2] << std::endl;
    }
    else {
        std::cout << 0;
    }

    return 0;
}

```

Дневник отладки

Достаточно долго думал над идеей решения этой задачи, так как первоначально придуманный мной алгоритм выдавал WA.

Тест производительности

Померить время работы кода лабораторной и теста производительности на разных объёмах входных данных. Сравнить результаты. Проверить, что рост времени работы при увеличении объема входных данных согласуется с заявленной сложностью.

Сложность по времени: в алгоритме имеется вложенный цикл, один из них проходит от 0 до $n - 2$, другой от $i + 1$ до $n - 1$, итоговая сложность будет $O(n^2)$.

Время работы алгоритма для разных n :

1. $n = 1000 \Rightarrow 0.010s$
2. $n = 10000 \Rightarrow 0.807s$
3. $n = 100000 \Rightarrow 77.470s$

Можно видеть, что при увеличении n в 10 раз скорость работы увеличивается примерно в 100 раз. Таким образом, можно сделать вывод, что моя асимптотическая оценка алгоритма абсолютно верна.

Недочёты

Возможно, можно было реализовать бинарный поиск для ускорения алгоритма.

Выводы

При выполнении данной лабораторной работы я ознакомился с жадными алгоритмами и смог решить задачу поиска треугольника с максимальной площадью по известным сторонам.