Лабораторная работа №8 по курсу Дискретного Анализа: Жадные алгоритмы

Выполнил студент группы 08-308 МАИ Жерлыгин Максим Андреевич

Условие

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.

Вариант 3:

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.

Метод решения

Рассмотрим поставленную задачу – требуется найти стороны треугольника из предложенных такие, что будут образовывать наибольшую площадь. Наивный алгоритм состоит в полном переборе всех сторон и выявления среди них треугольника с наболшей площадью. Сложность такого алгоритма $O(n^3)$ так, как кол-во ребер n на первом ребре можно взять из них n, вторыми n-1 и третьими n-2.

Удобно будет реализовать жадный алгоритм, так как исходя из задачи очевидно, что наибольшую площадь образуют стороны с наибольшим периметром. Для этого сортировкой подсчетом отсортируем ребра и применим вышеизложенный алгоритм (останавливаемся тогда, когда первый раз можем получить треугольник), однако, исходя из

теоремы, что наибольшую площадь образуют стороны равные друг другу, возможен такой исход, что ребра с наибольшим периметром не образуют треугольник с наибольшей площадью, а есть такой набор ребер, чей периметр будет равен, найденному набору ребер при прямом обходе. Для этого, после того, как найдем несколько ребер с наибольшим периметром в отсортированном массиве найдем периметры троек ребер и сравним их с полученным результатом, найдем максимум из этого множества (результат при наивном обходе + результаты подсчета троек ребер) и получим ответ.

Асимптотика данного решения в худшем случае $O(n^3)$, однако зачастую при большом наборе различных ребер скорость выполнения данного алгоритма будет стремится к $O(n)$, так как после сортировки среди первых троек будет находится нужная комбинация ребер. Объем затраченной памяти - $O(n)$.

Исходный код

Основными этапами здесь являются поиск троек с наибольшим периметром,

```
    for (long long i = sides.size() - 1; i >= 2; --i) {
        for (long long j = i - 1; j >= 1; --j) {
            for (long long k = j - 1; k >= 0; --k) {
                if (IsTriangle(sides[i], sides[j], sides[k])) {
                    first = sides[k];
                    second = sides[j];
                    third = sides[i];
                    Square = SquareOfTriangle(sides[i], sides[j], sides
                        [k]);
                    isFound = true;
                    break;
                }
            }
            if (isFound) {
                break;
            }
        }
        if (isFound) {
            break;
        }
    }
```

и поиск троек с равными сторонами.

```
    for (int i = sides.size() - 1; i >= 2; --i) {
        if (IsTriangle(sides[i], sides[i - 1], sides[i - 2])) {
            if (Square < SquareOfTriangle(sides[i], sides[i - 1], sides
                [i - 2])) {
                Square = SquareOfTriangle(sides[i], sides[i - 1], sides
                    [i - 2]);
                first = sides[i - 2];
                second = sides[i - 1];
                third = sides[i];
            }
        }
    }
```

Пример работы

```
mmaxim2710@DESKTOP-RDPBU3D:/mnt/c/Users/mmaxi/Desktop/coursera/DA_ex/
    lab8$ make clean
rm -f *.o solution
mmaxim2710@DESKTOP-RDPBU3D:/mnt/c/Users/mmaxi/Desktop/coursera/DA_ex/
    lab8$ make
g++ -std=c++17 -pedantic -Wall -O2 -c main.cpp -o main.o
g++ -std=c++17 -pedantic -Wall -O2 main.o -o solution
mmaxim2710@DESKTOP-RDPBU3D:/mnt/c/Users/mmaxi/Desktop/coursera/DA_ex/
    lab8$ ./solution
5
8 4 2 7 5
17.321
5 7 8
mmaxim2710@DESKTOP-RDPBU3D:/mnt/c/Users/mmaxi/Desktop/coursera/DA_ex/
    lab8$ ./solution
4
1 2 3 5
0
```

Вывод

Жадные алгоритмы в некоторых случаях позволяют оптимизировать наивное решение, но только в тех задачах, где важна максимальная выгода в текущий момент времени, если использовать жадные алгоритмы для решения задач, где результат зависит от правильного выбора или решения на всех шагах, то жадные алгоритмы не подойдут для решения.

Для многих задач жадные алгоритмы не дают приемлемого результата. К примеру, задача о рюкзаке: вор пробрался на склад, в котором хранятся три вещи весом 10 кг, 20 кг и 30 кг и стоимостью 60, 100 и 120 рублей соответственно. Вор максимум может унести 50 кг. Нужно максимизировать прибыль вора. Если поступать здесь жадно и выбирать самую ценную вещь(то есть, 6 рублей за кг первой штуки, 5 рублей за кг второй и 4 рубля за кг третьей), то вор должен взять первую вещь, потом останется место для второй вещи, однако оптимальное решение составляет вторая и третья вещь. Жадный алгоритм может показать решение, приближенное к оптимальному. Однако для большинства NP-полных задач, не всегда необходимо получать точное решение, так как решения, близкие к оптимальным, могут применяться в прикладных задачах.

Из этого следует вывод, что для жадных алгоритмов существует своя область применения.

Так же, глядя на решение моей задачи, а именно на 3 вложенных цикла, хочется сразу же дать асимптотику $O(n^3)$, однако после сортировки вектора отрезков нужная комбинация будет лежать среди последних трёх, и скорость будет стремиться к $O(n)$.