Лабораторная работа №7 по курсу дискретного анализа: Жадные алгоритмы

Выполнил студент группы М8О-309Б-23 Кривошапкин Егор.

Условие

Разработать жадный алгоритм решения задачи, определяемой своим вариантом. Доказать его корректность, оценить сложность алгоритма.

Реализовать программу на языке C или C++, соответствующую 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. 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

Решение задачи происходит следующим образом: изначально все длины сторон треугольников сортируются по убыванию (благодаря этому при проходе по массиву длин мы будем получать треугольники с максимальными длинами сторон). Далее выполняется обход массива длин. Для каждой тройки длин проверяется неравенство треугольника, а затем, если тройка сторон образует треугольник, вычисляется площадь данного треугольника по формуле Герона и сохраняется в том случае, если она больше сохранённой ранее максимальной площади.

Докажем, что такой метод решения приведёт нас к верному ответу. После сортировки длин сторон по убыванию, мы получим массив a из N элементов, в котором:

$$a_1 \geq a_2 \geq a_3 \geq \cdots \geq a_N$$

Предположим, что мы взяли тройку элементов a_i , a_j , a_k , $i<j<k$ (тройку не подряд идущих элементов этого массива) и тройку элементов a_i , a_{i+1} , a_{i+2} (ближайшую к i подряд идущую тройку элементов этого массива). Заметим, что, так как наш массив сторон отсортирован по убыванию, для любых выбранных таким способом элементов массива выполняются условия: $a_i = a_i$, $a_{i+1} >= a_j$, $a_{i+2} >= a_k$. Как видим, длины сторон треугольника подряд идущих троек всегда как минимум не меньше, чем длины сторон треугольника не подряд идущих выбранных троек. Из формулы Герона для вычисления площади заметим: чем больше сумма длин сторон треугольника, тем больше будет его площадь (так как площадь напрямую зависит от полупериметра), следовательно, мы получаем стороны, образующие треугольник с максимально возможной площадью.

В итоге мы получили «жадный» алгоритм (на каждом шаге выбираем треугольник с максимально возможными сторонами) для решения поставленной задачи.

Описание программы

Мой основной файл main.cpp состоит из вспомогательных функций triangleBuildable, calculateArea и основной функции int main():

triangleBuildable – проверка возможности построить треугольник из заданных сторон с помощью неравенства треугольника

calculateArea – вычисление площади треугольника по формуле Герона

Int main() – считывание длин сторон треугольника, сортировка по убыванию, перебор троек с сохранением максимальной найденной площади.

Дневник отладки

Во время реализации я столкнулся с двумя ошибками:

1) 20.10.25 19:03:59 WA на 4 тесте.

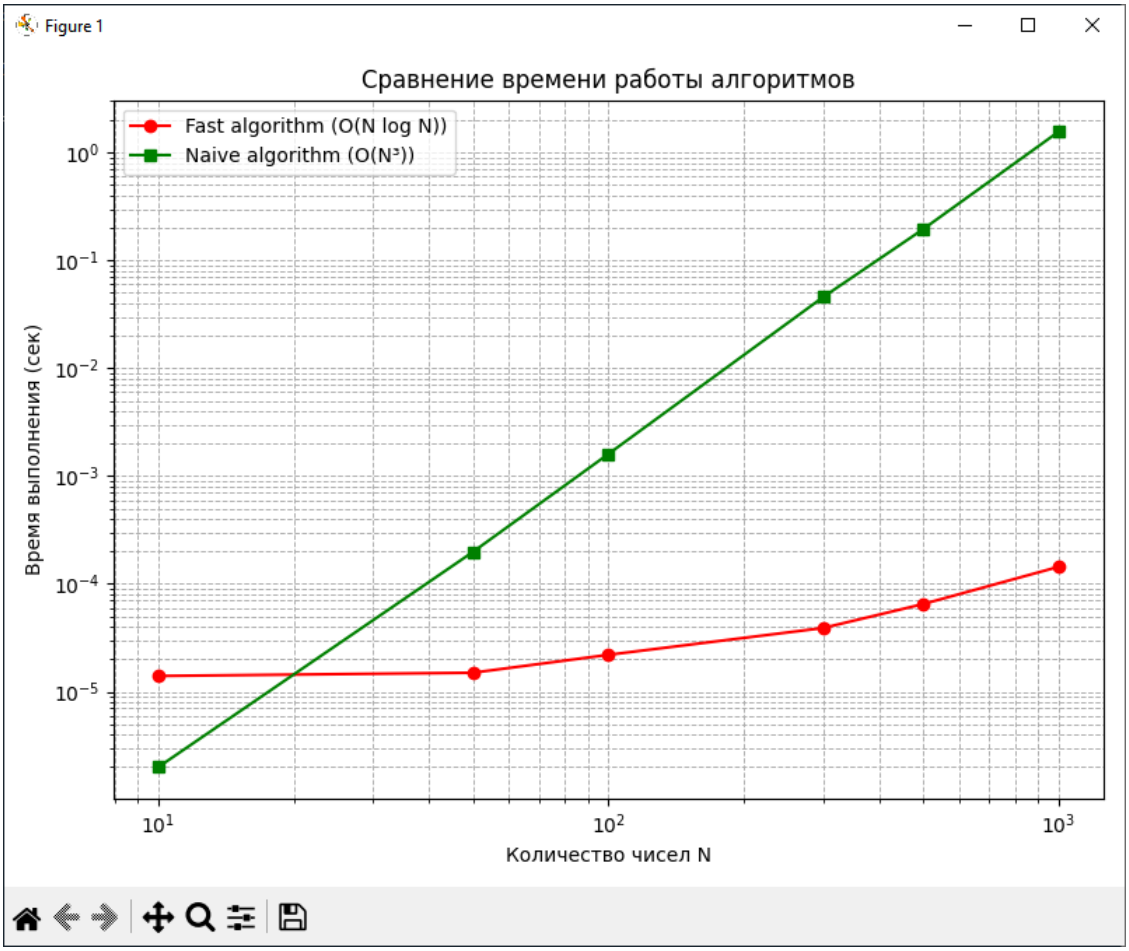
Ошибка заключалась в том, что я пытался выбрать подходящие стороны без записи всех вариантов длин в массив, а по ходу их поступления. Разумеется, данное решение увенчалось провалом, так как алгоритм пропускал очень много потенциально успешных комбинаций.

2) 20.10.25 19:07:42 WA на 9 тесте.

Ошибка заключалась в том, что при обходе отсортированного массива длин я выводил площадь первого же треугольника, стороны которого удовлетворяли неравенству треугольника, при этом я не учел, что в тестах могут быть записаны такие длины сторон, треугольник из которых получается вырожденным, а площадь такого треугольника равна нулю.

Тест производительности

Для оценки скорости работы моего алгоритма я сравнил его с наивным алгоритмом решения данной задачи (перебором всех троек за $O(n^3)$). Скорость работы программы проверялась на следующих длинах массива сторон: 10, 50, 100, 300, 500, 1000.



Как видим, жадный алгоритм решения задачи намного превосходит наивный, что неудивительно, ведь сложность алгоритма равна O(n*logn), что значительно превосходит кубическую сложность решения.

Докажем, что жадный алгоритм работает за O(n*logn). Вначале производится сортировка массива длин по убыванию с помощью std::sort и std::reverse – это O(n*logn). Далее выполняется один проход по массиву, параллельно выполняя действия, которые производятся за константную сложность – это O(n). Итого получим: O(n*logn + n) = O(n*logn).

Выводы

Выполнив данную лабораторную работу, я научился применять идею «жадного» программирования для решения задачи. Мне крайне понравился данный подход, так как он даёт значительное преимущество перед наивным алгоритмом и требует достаточно мало усилий при разработке идеи и написания кода. Данный подход, несомненно, улучшил мои навыки как программиста и поможет мне в будущем при решении алгоритмических задач.