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: А.Д. Волков
Преподаватель: А. А. Кухтичев
Группа: М8О-306Б

Дата: 17.11.2024
Оценка:
Подпись:

Москва, 2024

Лабораторная работа №8

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

1 Описание

Требуется реализовать жадный алгоритм для вычисления максимально возможной площади треугольника, построенного из определенных длин сторон. Давайте дадим определение жадным алгоритмам.
Жадные алгоритмы — это класс алгоритмов, которые принимают решения, основываясь на текущем состоянии задачи, выбирая наилучший (или наиболее выгодный) вариант на каждом шаге, в надежде, что он приведет к оптимальному решению глобальной задачи. Жадные алгоритмы не всегда приводят к оптимальным решениям, но во многих задачах дают нужный результат. [1]

2 Исходный код

Сначала мы считываем все данные, и стороны треугольников помещаем в вектор. Затем, сортируем наш вектор по убыванию величин сторон. Этот шаг становится очевидным, если учесть, что у треугольника с наибольшими сторонами, площадь будет наибольшая. Затем запускаем цикл, в котором мы берем три стороны подряд и по ним вычисляем площадь, перед этим проверив неравенство треугольника. Затем мы просто храним эту площадь и идем дальше по циклу. После выполнения цикла у нас есть правильный ответ.

```
1  #include <bits/stdc++.h>
2
3  // triangle class
4  class TTriangle
5  {
6  public:
7      int a, b, c;
8      // sides constructor
9      TTriangle(int a, int b, int c)
10     {
11         this->a = a;
12         this->b = b;
13         this->c = c;
14     }
15     // triangle are with heron formula
16     double HeronArea()
17     {
18         double perimetr = (1.0 * (this->a + this->b + this->c)) / 2;
19         return sqrt(perimetr * (perimetr - this->a) * (perimetr - this->b) * (perimetr
               - this->c));
20     }
21 };
22
23 int main()
24 {
25     int n;
26     std::cin >> n;
27     std::vector<int> side_sizes;
28     for (int _ = 0; _ < n; ++_)
29     {
30         int input;
31         std::cin >> input;
32         side_sizes.push_back(input);
33     }
34     // reverse sorting
35     std::sort(side_sizes.begin(), side_sizes.end(), std::greater<>());
36     double max_area = 0;
37     int max_a = 0, max_b = 0, max_c = 0;
```

```
38      for (size_t i = 0; i < n - 2; ++i)
39      {
40          // rule of triangle existance
41          if (side_sizes[i] < side_sizes[i + 1] + side_sizes[i + 2])
42          {
43              TTriangle triangle(side_sizes[i], side_sizes[i + 1], side_sizes[i + 2]);
44              if (triangle.HeronArea() > max_area)
45              {
46                  max_area = triangle.HeronArea();
47                  max_a = triangle.a;
48                  max_b = triangle.b;
49                  max_c = triangle.c;
50              }
51          }
52      }
53      if (max_area > 0)
54      {
55          std::cout << std::fixed << std::setprecision(3) << max_area << "\n";
56          std::cout << std::defaultfloat << max_c << " " << max_b << " " << max_a << "\n"
                ;
57      }
58      else
59      {
60          std::cout << 0 << "\n";
61      }
62  }
```

main.cpp	
double HeronArea()	Метод для вычисления площади треугольника по формуле Герона
int main()	Точка входа программы

```
1   class TTriangle
2   {
3   public:
4       int a, b, c;
5       // sides constructor
6       TTriangle(int a, int b, int c);
7       // triangle are with heron formula
8       double HeronArea();
9   };
```

3 Консоль

```
lexasy@lexasy$ cat test.txt
5
1
2
3
4
5lexasy@lexasy$ make
g++ main.cpp -o solution
lexasy@lexasy$ ./solution <test.txt
6.000
3 4 5
```

4 Тест производительности

Тест производительности представляет из себя сравнение времени работы нашего алгоритма и наивного алгоритма, работающего за кубическое время на большом тесте, а именно на 671 стороне.

```
lexasy@lexasy$ make
g++ main.cpp -o solution
lexasy@lexasy$ make benchmark
g++ benchmark.cpp -o benchmark
lexasy@lexasy$ ./solution <test.txt | grep time
time: 365ms
lexasy@lexasy$ ./benchmark <test.txt | grep time
time: 402127ms
```

Как мы видим, скорость работы выше более чем в 1000 раз. Это объяснимо тем, что наш алгоритм работает за линейно-логарифмическое время и он естественно должен опережать наивный алгоритм, особенно на больших тестах.

5 Выводы

Выполнив лабораторную работу №8, я узнал как решать сложные задачи с помощью жадных алгоритмах. С помощью жадных алгоритмов можно сильно ускорять свои программы, что безусловно пригодиться в моей дальнейшей деятельности.

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))