Отчет по лабораторной работе №1

по курсу «Алгоритмика»

Выполнил:

Григорьев Илья

517 группа

Москва, 2022

Постановка задачи

На плоскости заданы два множества точек: B (черные) и W (белые) по n точек в каждом. Никакие три точки не лежат на одной прямой. Правильным паросочетанием называется множество из n пар (b,w), b ∈ B, w ∈ W такое, что отрезки прямых, соответствующие этим парам, не пересекаются. Требуется разработать и реализовать алгоритм, позволяющий по входным данным построить правильное паросочетание за время $O(n^2 logn)$. Необязательный критерий – предпочтительным является решение, минимизирующее среднюю длину пути между точками в парах.

Исходные данные поступают из файла построчно. В первой строке задано число последующих строк. В каждой последующей строке описана одна точка: id точки, координаты точки и цвет точки (0 или 1).

Выходные данные выводятся на консоль и записываются в файл (опционально). Формат выходных данных: в первой строке число найденных пар точек, во второй строке среднее расстояние между парами точек. Далее следуют n строк, в каждой строке – найденная пара точек (id двух точек).

Описание алгоритма

Для начала докажем, что всегда существует прямая, проходящая через черную и белую точки, для которой количество черных точек, попавших в одну из полуплоскостей относительно этой прямой, равно количеству белых точек в этой же полуплоскости.

Найдем самую нижнюю точку и будем рассматривать лучи, выходящие из нее. Все остальные точки сосредоточены в верхней полуплоскости, относительно этой точки. Будем «заметать» верхнюю полуплоскость лучами из выбранной точки по часовой стрелке (направление абсолютно не важно). Так как по условию никакие три точки не лежат на одной прямой, то можно выстроить оставшиеся 2n-1 точек в порядке «заметания» полуплоскости. Очевидно, что перебирая эти точки в выбранном порядке, найдется точка противоположного цвета такая, что среди пройденных точек равное количество черных и белых. Нижняя и найденная точки задают прямую. По построению все пройденные точки лежат в одной из полуплоскостей от этой прямой, а значит и по другую сторону от прямой лежит равное число черных и белых точек.

Опишем алгоритм нахождения двух точек, задающих такую прямую:

1. Находим нижнюю точку в массиве. $O(n)$
2. Вычисляем косинусы углов между векторами, задающими направления от нижней точки до всех остальных и единичным ортом (1, 0). $O(n)$
3. Сортируем массив точек по найденным косинусам углов. $O(n\,logn)$
4. Находим нужную точку за один проход по отсортированному массиву. $O(n)$

Мы описали алгоритм нахождения разделяющей прямой за $O(n\,logn)$, теперь вернемся к исходной задаче, которая почти решена. Опишем общий алгоритм решения задачи:

1. Находим пару точек, задающих разделяющую прямую. Для минимизации среднего расстояния используется модифицированный алгоритм, который находит ближайшую по расстоянию подходящую точку для нижней точки. Сложность остается $O(n\,logn)$.
2. Добавляем эту пару точек в результирующий массив, прибавляем расстояние между этими точками к суммирующей переменной.
3. Рекурсивно запускаем функцию от двух множеств точек, по разным сторонам от найденной прямой.
4. В конце делим сумму расстояний на число пар, чтобы найти среднее расстояние.

В предложенном решении используется парадигма «Разделяй и властвуй». На каждом этапе задача сводится к решению двух более простых подзадач. Решая эти подзадачи независимо друг от друга, мы получаем корректное решение общей задачи. Итоговая сложность алгоритма $O(n^2 logn)$, потому что задача нахождения разделяющей прямой решается n раз (находит n пар среди 2n точек).

Особенности реализации

Основная программа была реализована на C++14 для достижения высокой производительности. Программа для визуализации была реализована на Python3 с применением библиотеки Matplotlib. Результат

работы основной программы на C++ передается в программу для визуализации через текстовый файл. Для запуска необходимо скомпилировать код на C++, запустить исполняемый файл, сгенерированный файл result.txt положить рядом с файлом visualizer.py, запустить скрипт visualizer.py.

Были произведены замеры реального времени работы программы на C++ с помощью библиотеки chrono. Ниже представлены результаты на всех тестовых данных:

- 3.txt --- 0.0034ms
- 3-2.txt --- 0.0037ms
- 33-1.txt --- 0.122ms
- 33-2.txt --- 0.0831ms
- 333.txt --- 1.901ms
- 3333_float.txt --- 24.6499ms

Коды программ, результаты работы на тестовых данных и визуализации найденных пар находятся в архиве.

Выводы

Был разработан очень эффективный алгоритм решения, на всех тестовых данных код отрабатывает мгновенно. Поставленная задача была успешно решена, все медведи нашли себе пары без пересекающихся маршрутов, а время решения даже для 3333 пар составило менее 25 миллисекунд.