

ФГБОУ ВО «НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ НИЖЕГОРОДСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. Н.И. ЛОБАЧЕВСКОГО»
Институт Информационных Технологий, Математики и Механики
Фундаментальная информатика и информационные технологии

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

«Поиск пары пересекающихся отрезков: наивный алгоритм и на
основе АВЛ-дерева»

Выполнил:

Кораблев Никита Денисович
Группа: 3821Б1ФИЗ

Проверил:

Уткин Герман Владимирович
кафедра: АГДМ

Нижний Новгород
2023

Содержание

1	Введение	2
2	Постановка задач	2
2.1	Наивный алгоритм	2
2.2	Эффективный алгоритм	2
3	Руководство пользователя	3
3.1	Использование готовых тестов	3
3.2	Вызов алгоритмов вручную	3
4	Руководство программиста	4
4.1	Структура проекта	4
5	Тестирование	5
5.1	Аппаратные характеристики	5
5.2	Результаты	5
5.2.1	Тест 1	5
5.2.2	Тест 2	6
5.2.3	Тест 3	6
5.2.4	Тест 4	7
6	Заключение	8
7	Список Литературы	9

1 Введение

В данной лабораторной работе будут рассмотрены несколько алгоритмов, решающих задачу поиска пары пересекающихся отрезков. Так же, на основе их реализации и последующего тестирования будет проведена оценка их сложности, а так же сильные и слабые стороны.

2 Постановка задач

Задано множество S , состоящее из n отрезков на плоскости. Каждый отрезок $s_i = S[i]$ ($i=1, 2, \dots, n$) задан координатами его концевых точек в декартовой системе координат. Требуется определить, есть ли среди заданных отрезков по крайней мере два пересекающихся. Если пересечение существует, то алгоритм должен выдать значение “истина” (“true”) и номера пересекающихся отрезков s_1 и s_2 , в противном случае - “ложь” (“false”).

2.1 Наивный алгоритм

Наивный алгоритм подразумевает перебор всех отрезков до тех пор, пока не будет найдено первое пересечение, или его не будет совсем.

2.2 Эффективный алгоритм

В этом алгоритме используется так называемый метод вертикальной заметающей прямой, движущейся в сторону возрастания абсциссы. В каждый момент времени заметающая прямая пересекает отрезки, которые образуют динамически меняющееся множество L . Отрезки в множестве L упорядочиваются по неубыванию ординат точек их пересечения с заметающей прямой. Множество L представляется АВЛ-деревом и модифицируется с помощью операций удаления и вставки элементов.

В алгоритме используются две операции $L.getPrev(s)$ и $L.getNext(s)$, позволяющие за время $O(\log|L|)$ определить непосредственно предшествующий отрезок и отрезок, непосредственно следующий за отрезком s в последовательности L . Если не существует отрезка, предшествующего отрезку s или следующего за ним, то процедуры $L.getPrev(s)$ и $L.getNext(s)$ выдают произвольный отрезок, заведомо не пересекающийся с отрезками из множества S . Это требование введено для того, чтобы не усложнять алгоритм лишними деталями.

Операция $L.insert(s)$ осуществляется в момент, когда заметающая прямая достигает левого конца отрезка s , а операция $L.remove(s)$, когда она достигает его правого конца. Обе эти операции осуществляются за время $O(\log|L|)$.

В алгоритме используется массив точек из $2n$ элементов, которые соответствуют концам отрезков. В каждом элементе представлены координаты точки, номер отрезка и информация о том, каким концом, левым или правым, для соответствующего отрезка является данная точка.

3 Руководство пользователя

3.1 Использование готовых тестов

Для запуска готовых тестов, необходимо в main.cpp файле импортировать библиотеку "Tests.h" и в теле main функции вызвать тесты: test1(fileDir), test2(fileDir), test3(fileDir), test4(fileDir), где fileDir - это полный путь до .txt файла, куда будет сохраняться время работы алгоритмов.

Пример записи данных в файл для тестов 1, 2 и 3:

```
1 23 5120 14
101 28 5039 7
201 36 4627 6
301 33 4471 3
401 42 4447 3
...
```

1ый столбец - количество отрезков в множестве, 2ой - время работы наивного алгоритма, 3ий - время сортировки массива точек, и 4ый - время работы эффективного алгоритма. Четвертый тест, в 1ый столбец запишет длину отрезков всего множества.

Также, для получения "настоящих" случайных значений, в начале файла следует написать: srand(time(NULL)).

3.2 Вызов алгоритмов вручную

Для использования алгоритмов без тестов, достаточно вызвать библиотеку "Segments.h". После чего, создайте объект для хранения отрезков - Segs s;, и заполните его отрезками. Теперь можно вызвать либо наивный алгоритм:

```
<multiple segments>.intersectionNaive();
```

и эффективный:

```
<multiple segments>.sortPoints(0, <multiple segments>.getPointLen()-1);
<multiple segments>.intersectionEffective();
```

Для создания точек, используйте:

```
Point <point name>(x, y);
```

Для создания отрезков, используйте:

```
Segment <segment name>(<point 1>, <point 2>);
```

Чтобы добавить отрезок в множество отрезков, используйте ТОЛЬКО:

```
<multiple segments>.pushBack(<segment name>);
```

4 Руководство программиста

4.1 Структура проекта

PSeg.h - прототипы точки вектора и сегмента, PSeg.cpp - их реализация.

Tree.h - прототип АВЛ-дерева на основе отрезков, Tree.cpp - его реализация.

Segments.h - прототип множества сегментов, Segments.cpp - его реализация.

Tests.h - прототип тестов для замера времени работы наивного и эффективного алгоритмов, Tests.cpp - их реализация.

main.cpp - запускаемый проект для экспериментов.

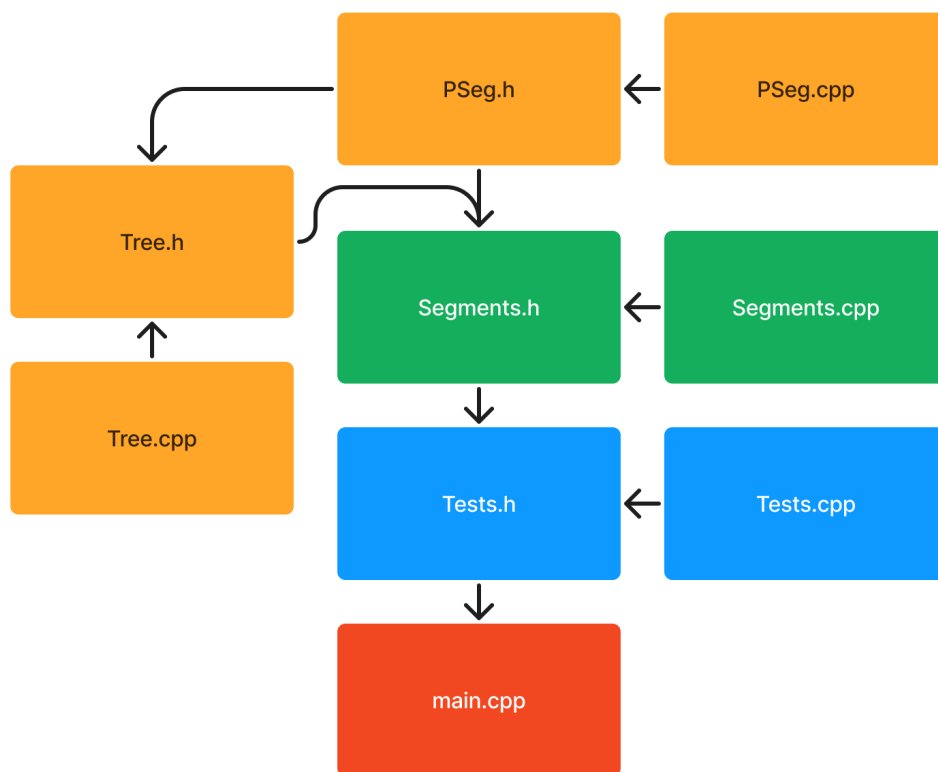


Рис. 1: Схема проекта

5 Тестирование

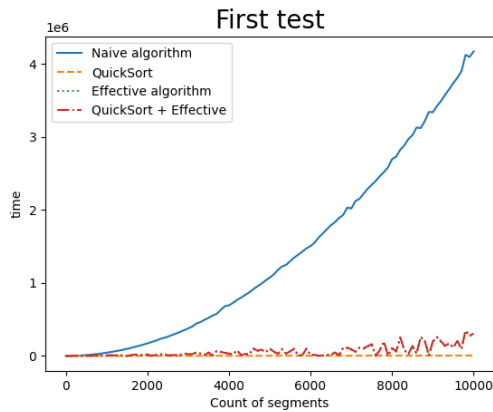
5.1 Аппаратные характеристики

- Ноутбук: Lenovo Legion 7 15IMH05
- Процессор: Intel® Core™ i7-10750H CPU @ 2.60GHz × 12 (разогнан до 4GHz)
- RAM: DDR4 - 2x 8ГБ
- OS: Ubuntu 22.04.3 LTS
- gcc: (Ubuntu 11.4.0-1ubuntu1-22.04) 11.4.0

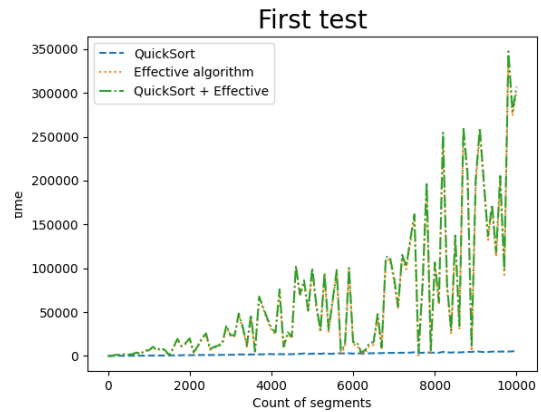
5.2 Результаты

5.2.1 Тест 1

Первый способ задания отрезков: $n = 1, \dots, 10^4$, с шагом 100.



(a) With naive algorithm



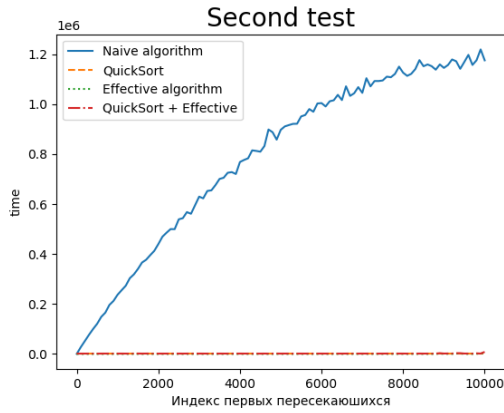
(b) Without naive algorithm

Рис. 2: Зависимость времени работы алгоритмов от количества не пересекающихся отрезков

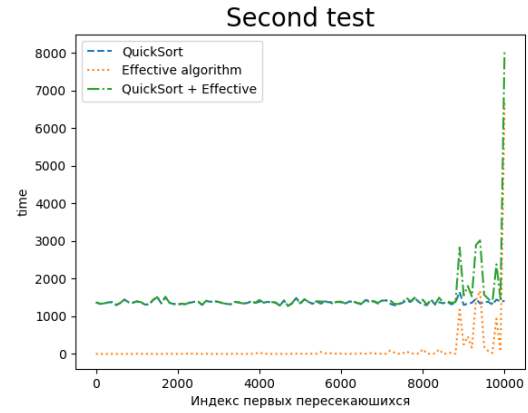
По сути, данный пример показывает худший вариант, когда во всем множестве отрезков нет ни одного пересечения, и сложность работы наивного алгоритма (Рис. 2) равна $O(n^2)$. Тем временем, график работы эффективного алгоритма показывает время работы $O(n \log(n))$.

5.2.2 Тест 2

Второй способ задания отрезков: $n = 10^4 + 3$, $k = 1, \dots, 10^4 + 1$, с шагом 100.



(a) With naive algorithm



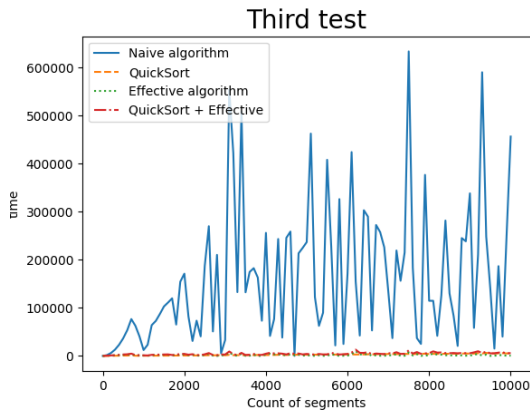
(b) Without naive algorithm

Рис. 3: Зависимость времени работы алгоритмов от места расположения 2ух пересекающихся отрезков.

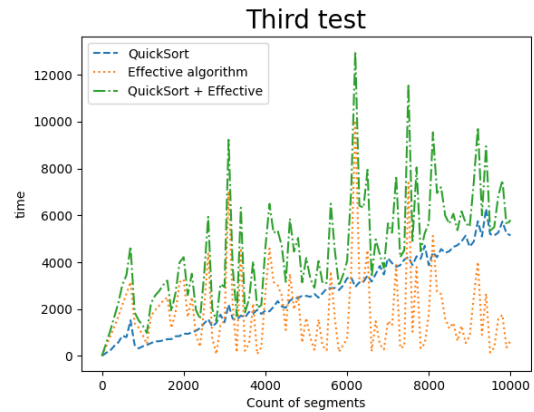
По скольку наивный алгоритм сравнивает попарно все отрезки в множестве, то время его работы напрямую зависит от расположения этих пересекающихся отрезков с индексами k и $k+1$. Но на время работы эффективного алгоритма, это ограничение не влияет.

5.2.3 Тест 3

Третий способ задания отрезков: $r = 0.001$, $n = 1, \dots, 10^4 + 1$, с шагом 100.



(a) With naive algorithm



(b) Without naive algorithm

Рис. 4: Зависимость времени работы алгоритмов числа отрезков размера r .

В отличие от первых двух тестов, где пересекающихся отрезков либо не было, либо их было 2, и они постепенно перемещались от начала множества к его концу, количество и расположение в множестве пересекающихся отрезков полностью случайно. Следовательно, мы получаем такой разброс времени работы наивного алгоритма.

5.2.4 Тест 4

Четвертый способ задания отрезков: $r = 1 * 10^{-4}, \dots, 0.01$ с шагом $1 * 10^{-4}$, $n=10^4$.

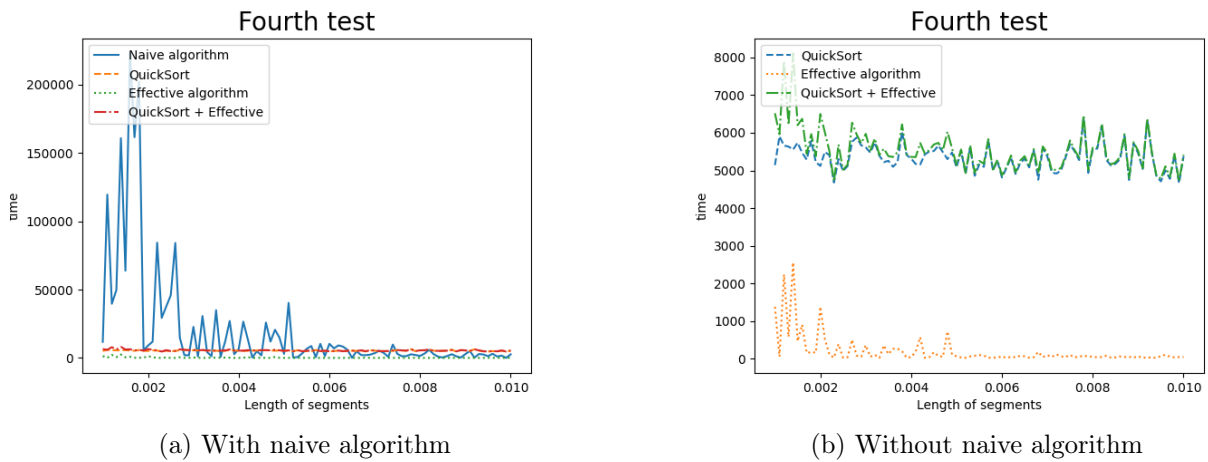


Рис. 5: Зависимость времени работы алгоритмов от размера отрезков.

Из графиков видно, что после достижения отрезками некоторой длины, время работы наивного алгоритма начинает уменьшаться. А эффективный алгоритм почти всегда работает за константное время. Так же можно заметить, что всплески на графике эффективного алгоритма совпадают с всплесками наивного.

6 Заключение

Можно сделать вывод, что стоит тратить усилия на разработку алгоритма эффективного поиска пересечений, только если мы имеем относительно большое множество отрезков малой длины (первые три теста). В остальных случаях, за счет простоты реализации, наивный алгоритм будет более эффективным. Так же, слабым местом эффективного алгоритма, является сортировка.

7 Список Литературы

- [1] Семинар 8. Метод заметающей прямой (Алгоритмы и структуры данных, часть 1). URL: https://www.youtube.com/watch?v=sINii2mwYls&t=481s&ab_channel=ComputerScienceCenter/
- [2] GeeksforGeeks. URL: <https://www.geeksforgeeks.org/introduction-to-avl-tree/?ref=lbp>
- [3] ХАБР. URL: <https://habr.com/ru/articles/267037/>
- [4] Моя программа. URL: https://github.com/NikitaKorablev/Intersections_of_segments