МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
Высшего образования
«Нижегородский государственный университет им. Н.И. Лобачевского»
Национальный исследовательский университет

Институт информационных технологий, математики и механики
Кафедра алгебры, геометрии и дискретной математики

ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ
«Поиск пары пересекающихся отрезков»

Выполнил:
студент группы 381706-1
Митягина Дарья Сергеевна

_____
(подпись)

Научный руководитель:
профф. каф. АГДМ ИИТММ,
Малышев Дмитрий Сергеевич

_____
(подпись)

Нижний Новгород
2019

Содержание

1. Введение

Хороший разработчик создает удобное и полезное программное обеспечение, а очень хороший старается сделать его максимально эффективным в использовании. Для этого необходимо понимать, почему и в каких пределах работает программа.

При разработке алгоритмов очень важно иметь возможность оценить ресурсы, необходимые для проведения вычислений, результатом оценки является функция **сложности (трудоемкости)**. Оцениваемым ресурсом чаще всего является процессорное время (*вычислительная сложность*) и память (*сложность алгоритма по памяти*). Оценка позволяет предсказать время выполнения и сравнивать эффективность алгоритмов.

В данной лабораторной работе будут рассмотрены алгоритмы поиска пары пересекающихся отрезков. Сравнение эффективного алгоритма и его примитивного аналога будет производится по временной сложности.

Цель работы : Исследовать вычислительную сложность алгоритма поиска пары пересекающихся отрезков.

2. Постановка задачи

Данная лабораторная работа включает в себя следующие подзадачи :

1. Выполнение реализации примитивного алгоритма, решающего поставленную задачу;
2. Разработка и реализация алгоритма с использованием AVL-деревьев ;
3. Проведение экспериментов, фиксирование результатов;
4. Сравнение теоретических оценок с полученными результатами.

3. Описание алгоритмов

Формулировка задачи : Задано множество S, состоящее из n отрезков на плоскости. Каждый отрезок si, (i = 1, 2,…, n) задан координатами его концевых точек в декартовой системе координат. Требуется определить, есть ли среди заданных отрезков по крайней мере два пересекающихся (см. [1], [3]). Если пересечение существует, то алгоритм должен выдать значение “истина” (“true”), и номера пересекающихся отрезов, в противном случае - “ложь” (“false”).

3.1. Наивный алгоритм поиска пересечений

Перебираются пары отрезков до тех пор, пока не будет обнаружено пересечение, либо же не будут исчерпаны все пары.

Псевдокод :

```
function ПРОВЕРКА_ПЕРЕСЕЧЕНИЯ_ОТРЕЗКОВ_1(S; n): boolean;
begin
b:= false; i:= 1;
while (i<n) and (not b) do begin
j:= i+1;
while (i≤n) and (not b) do begin
if (s[i] пересекает s[j]) then b:= true;
j:= j+1;
end;
i:= i+1;
end;
ПРОВЕРКА_ПЕРЕСЕЧЕНИЯ_ОТРЕЗКОВ_1:= b;
end;
```

Временная сложность этого наивного алгоритма O(n2).

Рассмотрим алгоритм более подробно. Реализация построена на основе следующих проверок:

1. Проверяем, что отрезки лежат в одной плоскости;
2. Проверяем, что пересекаются их ограничивающие параллелепипеды;
3. Для каждого из двух отрезков проверяем, что его концы лежат по разные стороны от прямой, проходящей через другой отрезок.

3.2. Эффективный алгоритм поиска пересечений

В этом алгоритме используется так называемой метод вертикальной заметающей прямой, движущейся в сторону возрастания абсцисс. В каждый момент времени заметающая прямая пересекает отрезки, которые образуют динамически меняющейся 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.

Псевдокод :

```
procedure intersection_effective (var b: Boolean; s1, s2: integer);
begin Лексикографическая сортировка массива ТОЧКА[1..2n]; b:= false;
for i:= 1 to 2*n do
begin p:= ТОЧКА[i]; s:= отрезок, концом которого является p;
if (p - левый конец отрезка s) then
begin ВСТАВИТЬ(s, L); s1:= НАД(s, L); s2:= ПОД(s, L);
if (s1 пересекает s) or (s2 пересекает s) then {b:= true; exit}
end else
begin s1:= НАД(s,L); s2:= ПОД(s, L);
if (s1 пересекает s2) then {b:= true; exit};
УДАЛИТЬ(s,L);
end;
end;
end;
```

Временная сложность данного нетривиального алгоритма проверки пересечения оценивается сверху величиной O(n×log n).

Мы используем неявные ключи, то есть поиск следуюшего и предыдущего элементов, а также их удаление, не используют самих ключей, а используют лишь структуру самого дерева ~ отсюда и логарифмическое время.

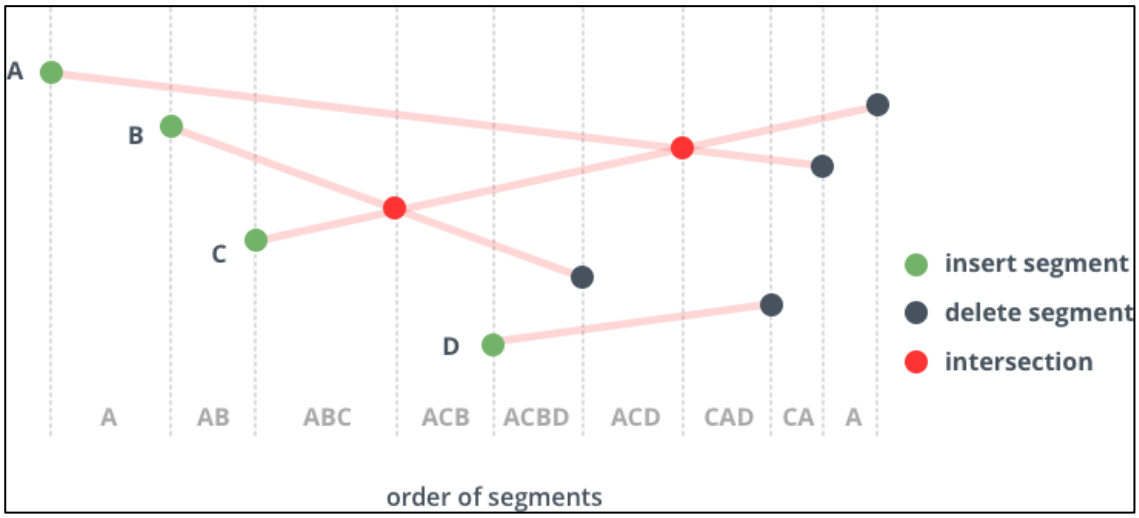

Рис. 0 – sweep line алгоритм (наглядный пример)

4. Руководство пользователя

При запуске программы пользователю будет представлен следующий интерфейс :

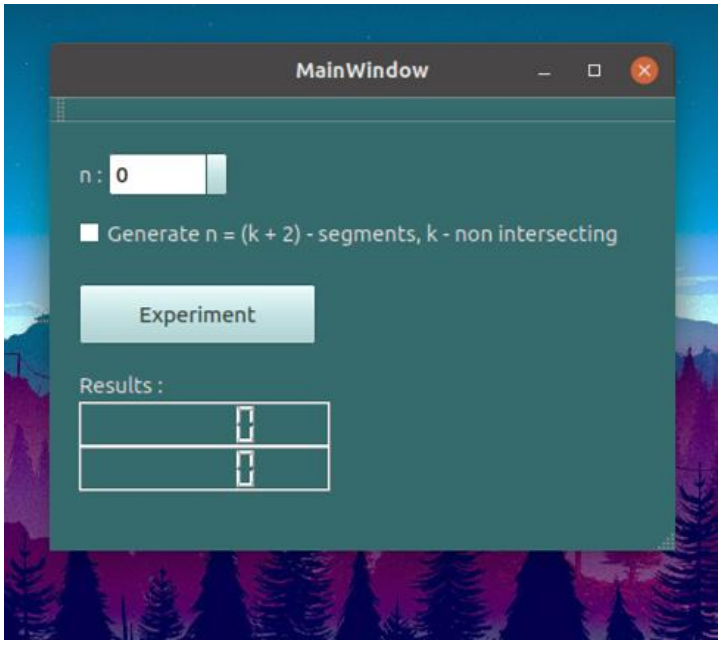

Рис. 1 – начало работы программы

Пользователю необходимо ввести количество отрезков. В данном случае сгенерируются (n – 2) непересекающихся отрезков и 2 пересекающихся, находящихся в конце списка сгенерированных отрезков.

Данный пример представляет собой худший, и именно поэтому наиболее наглядный для оценки временной сложности алгоритмов, случай.

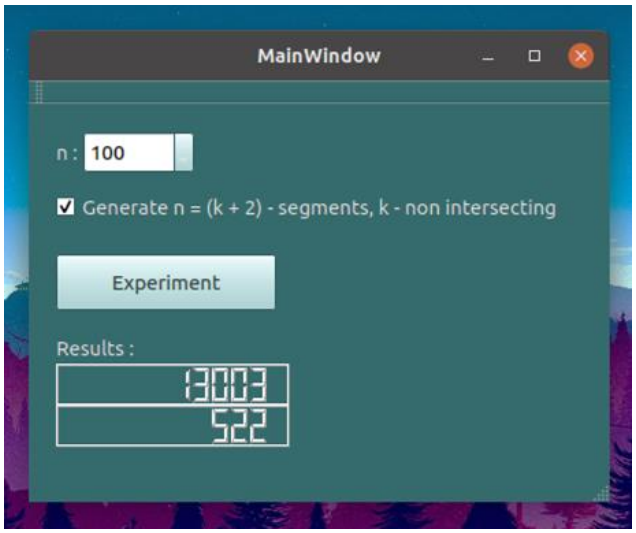

Рис. 2 – эксперимент (100 отрезков)

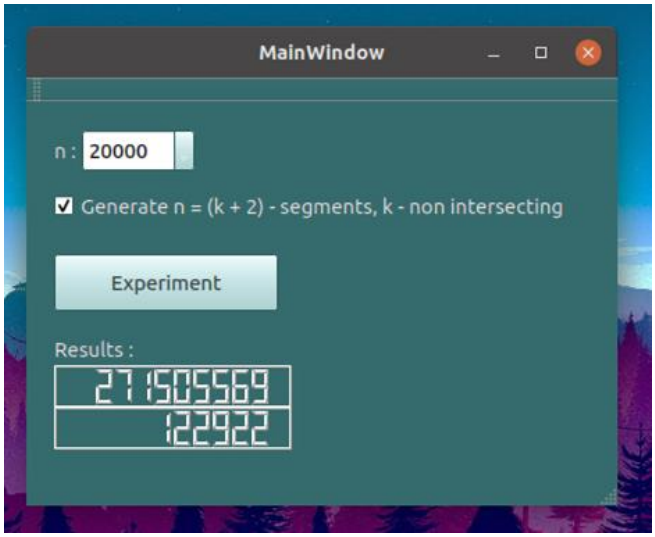

Рис. 3 - эксперимент (20 тыс. отрезков)

5. Описание программы

5.1. Структура программы

Данный проект сосдоит из следующих модулей :

1. Функциональная часть

    1.1. avl_tree.cpp – реализация структуры avl-дерева

    1.2. intersect.cpp – наивный алгоритм поиска пересечений

    1.3. line_segment.cpp – класс отрезок

    1.4. my_vector.h, my_vector.cpp

    1.5. sweep_line.cpp

    1.6. segments_generate.cpp

2. Интерфейс

    2.1. mainwindow.cpp

    2.2. mainwindow.ui

    2.3. main_work.cpp

5.2. Структуры данных

АВЛ-дерево— сбалансированное по высоте двоичное дерево поиска: для каждой его вершины высота её двух поддеревьев различается не более чем на 1.

АВЛ— аббревиатура, образованная первыми буквами фамилий создателей (советских учёных) Георгия Максимовича Адельсон-Вельского и Евгения Михайловича Ландиса.

Основные операции и их сложность:

Операция	Средний случай (average case)	Худший случай (worst case)
Add(*key, value*)	$O(\log n)$	$O(\log n)$
Lookup(*key*)	$O(\log n)$	$O(\log n)$
Remove(*key*)	$O(\log n)$	$O(\log n)$
Min	$O(\log n)$	$O(\log n)$
Max	$O(\log n)$	$O(\log n)$

Рис. 4 – оценка сложности операций

Относительно АВЛ-дерева балансировкой вершины называется операция, которая в случае разницы высот левого и правого поддеревьев = 2, изменяет связи предок-потомок в поддереве данной вершины так, что разница становится <= 1, иначе ничего не меняет. Указанный результат получается вращениями поддерева данной вершины.

Используются 4 типа вращений:

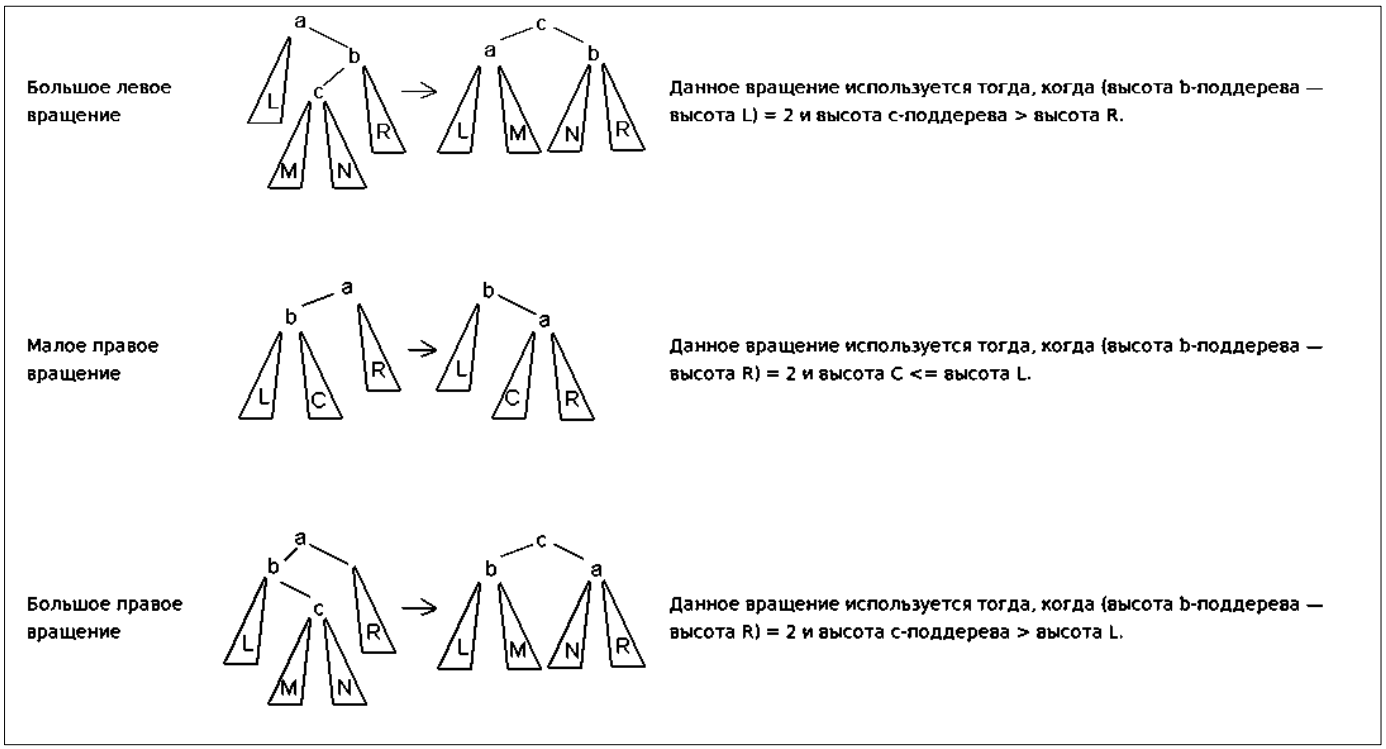

Рис. 5 – типы вращений при балансировке

6. Эксперименты

Эксперименты проводились на ПК с следующими параметрами:
Intel® Pentium(R) CPU N3710 @ 1.60GHz × 4
Ubuntu 18.10
64-bit

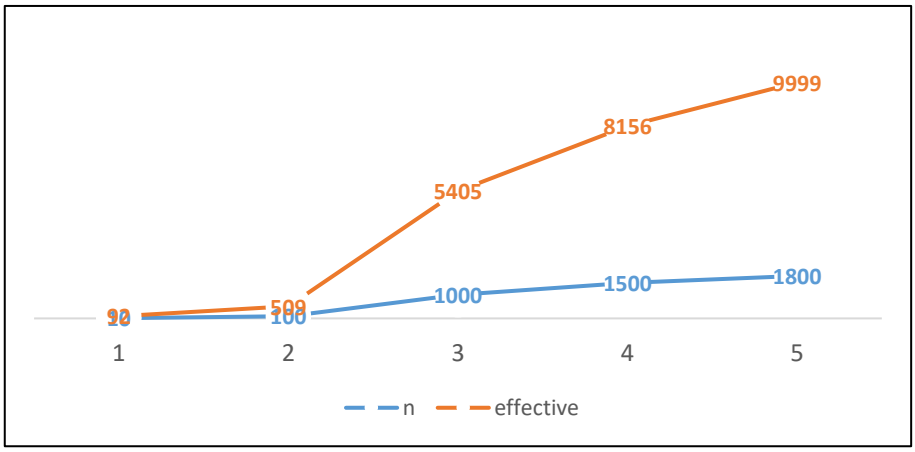

Рис. 6 – результаты экспериментов (эффективный алгоритм)

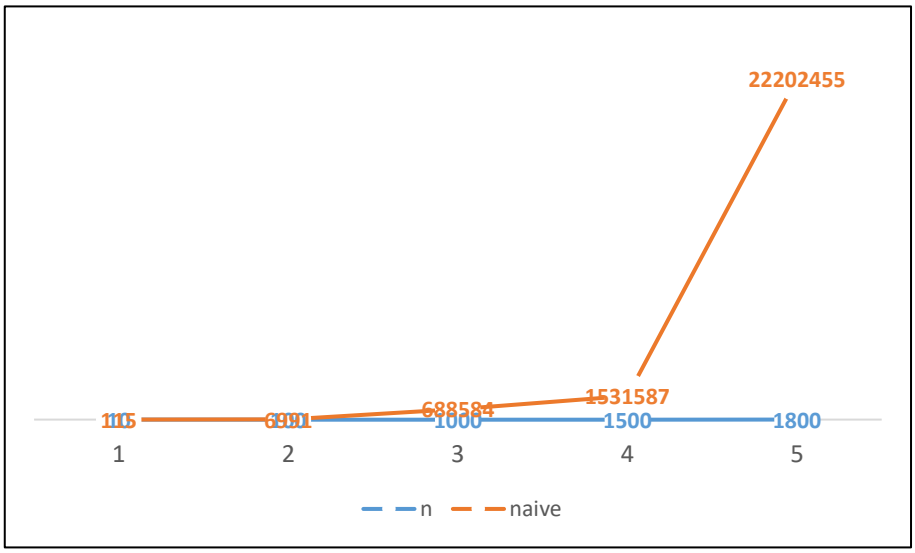

Рис. 7 – результаты экспериментов (наивный алгоритм)

n	Primitive(real)	Effective(real)
10	115	92
100	6991	509
1000	688584	5405
1500	1531587	8156
1800	22202455	9999

Таблица 1 – результаты экспериментов

n	Primitive(theory)	Effective(theory)
10	100	40
100	10000	664
1000	1000000	9965
1500	2250000	15825
1800	3240000	19458

Таблица 2 – теоретические предположения

7. Заключение

В ходе выполнения лабораторной работы была разработана и реализована программа, выполняющая алгоритм поиска пары пересекающихся отрезков. Были проведены вычислительные эксперименты над реализованными алгоритмами. Эксперименты показали, что наивный алгоритм уступает по скорости алгоритму, использующему АВЛ-деревья, как в теории, так и на практике.

Так же был получен навык работы с ранее мало знакомой структурой данных, как АВЛ-дерево.

8. Литература

Интернет ресурсы :

1. https://ru.wikipedia.org/wiki/%D0%90%D0%92%D0%9B-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE
2. https://habr.com/ru/post/150732/
3. https://en.wikipedia.org/wiki/Sweep_line_algorithm
4. https://www.topcoder.com/community/competitive-programming/tutorials/line-sweep-algorithms/
5. https://courses.csail.mit.edu/6.006/spring11/lectures/lec24.pdf