



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное автономное образовательное учреждение высшего образования  
**«Дальневосточный федеральный университет»**  
(ДВФУ)

---

**ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ  
ТЕХНОЛОГИЙ**

**Департамент математического и компьютерного моделирования**

**ДОКЛАД**

о практическом задании по дисциплине «АИСД»  
«Дерево интервалов»

**НАПРАВЛЕНИЕ ПОДГОТОВКИ**

09.03.03 «Прикладная информатика»  
«Прикладная информатика в компьютерном дизайне»

Выполнил студент  
гр. Б9121-09.03.03 пикд  
Чурганов Никита Сергеевич

\_\_\_\_\_  
(подпись)

Руководитель  
Доцент ИМКТ А.С Кленин  
(должность, уч. звание)

\_\_\_\_\_  
(подпись)

« \_\_\_\_ » \_\_\_\_\_ 2022г.

Доклад защищен:  
С оценкой \_\_\_\_\_

Рег. № \_\_\_\_\_

« \_\_\_\_ » \_\_\_\_\_ 2022 г.

г. Владивосток  
2022г

## Оглавление

Глоссарий .....	3
Введение.....	4
1 История развития .....	6
2 Суть алгоритма .....	7
3 Построение дерева интервалов.....	8
4 Операции над деревом.....	10
4.1 Добавление.....	10
4.2 Проверка на перекрытие .....	10
4.3 Удаление узла.....	13
5 Формальная постановка задачи .....	15
6 Реализация .....	16
6.1 Добавление.....	16
6.2 Поиск перекрываемых интервалов .....	17
6.3 Удаление .....	18
7 Тестирование .....	19
8 Исследование.....	22
8.1 Добавления узла в дерево.....	22
8.2 Анализ удаления узла из дерева .....	23
8.3 Анализ удаления узла из дерева .....	23
8.4 Вывод.....	24
<b>Список литературы .....</b>	<b>25</b>

## Глоссарий

**Коллекция** в программировании — программный объект, содержащий в себе, тем или иным образом, набор значений одного или различных типов, и позволяющий обращаться к этим значениям. [23]

**Метод** — это блок кода, содержащий ряд инструкций. Программа инициализирует выполнение инструкций, вызывая метод и указывая все аргументы, необходимые для этого метода. [22]

**Лист** — узел, не имеющий узлов-потомков на дереве. [21]

**Поддерево** — часть древообразной структуры данных, которая может быть представлена в виде отдельного дерева. [21]

**Корневой узел** — узел, не имеющий предков (самый верхний). [21]

**Корень** — одна из вершин, по желанию наблюдателя. [21]

**Интервал** — множество всех чисел, удовлетворяющих строгому неравенству  $a < x < b$ . [25]

## Введение

### *Суть и назначение*

Интервальное дерево – это древовидная структура данных, узел которой представляет интервал и максимальное значение в поддереве, поэтому характеризуется начальным и конечным значениями. В частности, дерево интервалов позволяет эффективно находить все интервалы, которые перекрываются любым заданным интервалом (точкой).

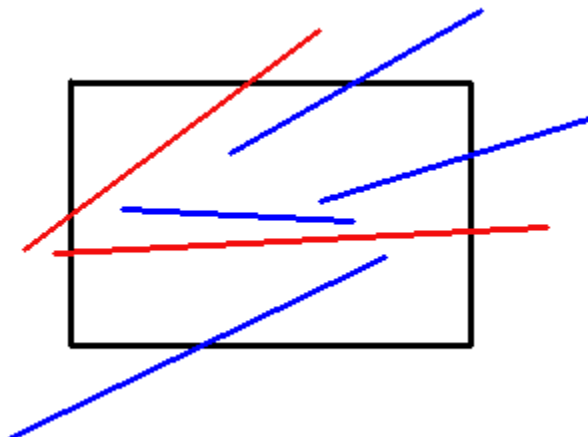
### *Пример использования*

Дерево интервалов используется для поиска видимых элементов внутри трехмерной сцены.

Имея набор отрезков и выровненный по оси прямоугольник, необходимо найти точки пересечения отрезков с прямоугольником. Эту проблему можно решить, используя дерево интервалов в сочетании с деревом диапазонов.[14]

Деревья диапазонов — это эффективная структура данных для поиска набора точек, присутствующих в диапазоне/прямоугольнике. Таким образом, их можно использовать для поиска всех сегментов линии, так что одна из конечных точек каждого сегмента линии присутствовала в прямоугольнике. Они соответствуют сегментам синей линии на рисунке.

Деревья интервалов можно использовать для поиска тех сегментов, которые пересекаются с прямоугольником, но конечные точки которых находятся за пределами окна. Это красные линии на рисунке.



Если рассмотреть ограниченную версию задачи, в которой все отрезки горизонтальны или вертикальны, то в таком случае любой горизонтальный отрезок, пересекающий прямоугольник, должен пересекать левый (правый) вертикальный край прямоугольника. Если представить горизонтальные сегменты как интервалы, а вертикальный край прямоугольника как точку, то проблема состоит в том, чтобы найти все интервалы, содержащие эту точку. Таким образом, задача решается, используя дерево интервалов. Точно так же находятся все пересекающиеся вертикальные отрезки.

## **1 История развития**

Точной информации в открытых источниках нет, но можно предположить, что дерево интервалов появилось примерно в 1979 г. Это можно обосновать тем, что бинарное дерево поиска появилось в 1960 г. и бинарное дерево поиска более простая структура данных в сравнении с деревом интервалов, так же есть такая структура данных, как дерево диапазонов, которая отдаленно похожа на дерево интервалов и появилось в 1979 г.

## 2 Суть алгоритма

Дерево интервалов позволяет решать следующую задачу. Дано множество отрезков  $I = \{[x_1, y_1], \dots, [x_n, y_n]\}$  и множество запросов. Каждый запрос характеризуется интервалом  $[q_x, q_y]$ . Для каждого запроса необходимо определить множество отрезков из  $I$ , которые перекрывают  $[q_x, q_y]$ .

Построение дерева интервалов занимает время  $O(n \log n)$ , а также  $O(n)$  памяти. На каждый запрос дерево интервалов позволяет отвечать за  $O(\log n + k)$ , где  $k$  – размер ответа на запрос.

### 3 Построение дерева интервалов

Изначально дерево не заполнено, поэтому берется произвольный интервал например,  $[5;10]$  и вставляется в дерево. Таким образом, дерево состоит из одного узла, который является корневым узлом (рисунок 1).



Рисунок 1 – Добавление первого узла

Затем берется следующий произвольный интервал  $[3; 12]$ , так как  $3 < 5$ , то переходим к левому сыну (рисунок 2).

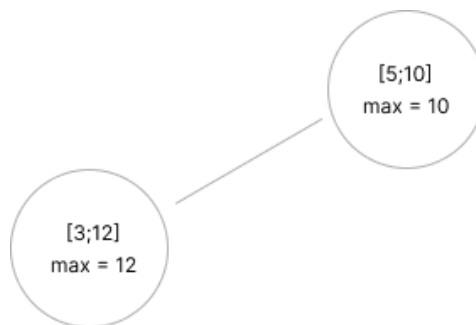


Рисунок 2 – Добавление узла в дерево

Далее проверяется максимум в левом поддереве и в правом при их наличии, если максимум в левом поддереве больше максимума в правом поддереве и максимума в корне, то в корень записывается максимум левого поддерева, если максимум в правом поддереве больше, то в корень записывается максимум правого поддерева, иначе в корень записывается самого корня (рисунок 3).



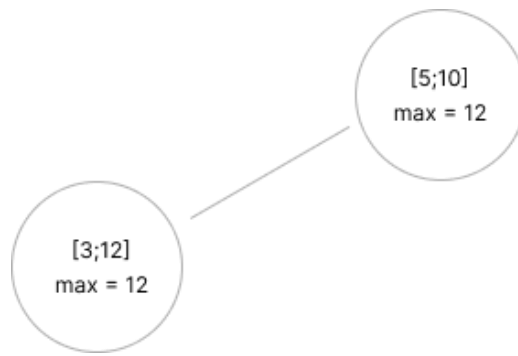


Рисунок 3 – Вычисление максимума в дереве

Аналогично предыдущим действиям добавляются другие узлы (рисунок 4).

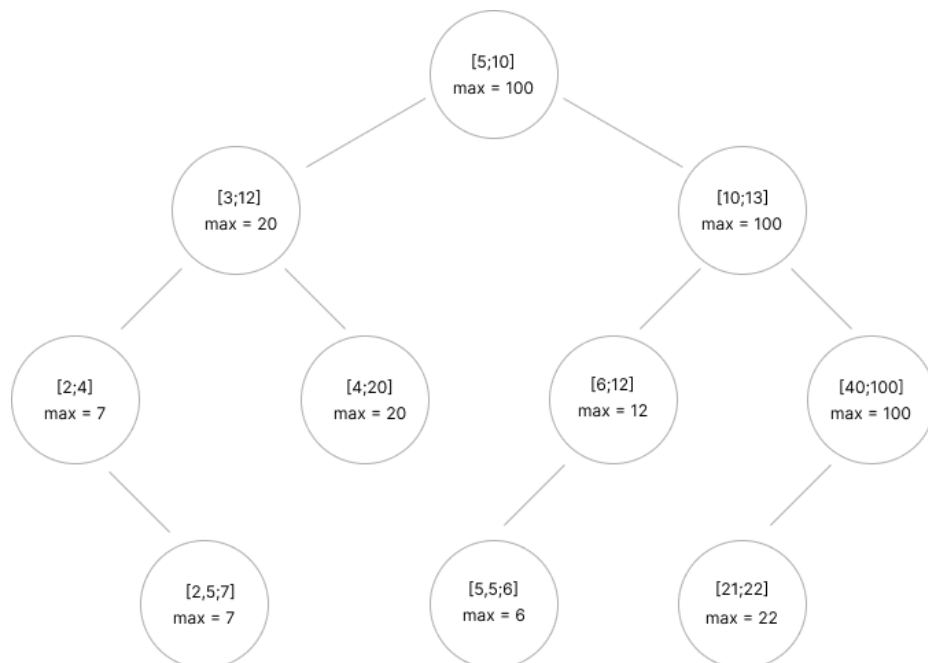


Рисунок 4 – Полученное дерево

## 4 Операции над деревом

1. Добавление
2. Проверка на перекрытие
3. Удаление

### 4.1 Добавление

Для добавления узла в дерево вводится интервал, который необходимо добавить, и если нижняя граница интервала меньше нижней границы интервала в корне или равна ей, то переходим к левому сыну, иначе – к правому. После того, как узел добавлен пересчитывается максимум, то есть если верхняя граница добавленного узла больше максимума в родительском узле, то максимум родительского узла принимает значение верхней границы добавленного интервала. Данные действия выполняются до тех пор, пока добавляемый узел не станет листом (рисунок 1 – рисунок 3).

### 4.2 Проверка на перекрытие

Для проверки на перекрытие вводится интервал  $x$ . В ходе прямого обхода выполняется сравнение границ интервала  $root$  текущего узла дерева и границ интервала  $x$ . Для корректного выполнения операции перекрытия дерево должно иметь хотя бы один узел.

Введём обозначения:

$x.low$  – нижняя граница интервала  $x$ ,

$x.high$  – верхняя граница интервала  $x$ ;

$root.low$  – нижняя граница интервала  $root$ ,

$root.high$  – верхняя граница интервала  $root$ .

Варианты перекрытия интервалов:

1. Интервал  $x$  частично перекрывает интервал  $root$ , при этом  $x$  выходит за правую границы  $root$  (рисунок 5).

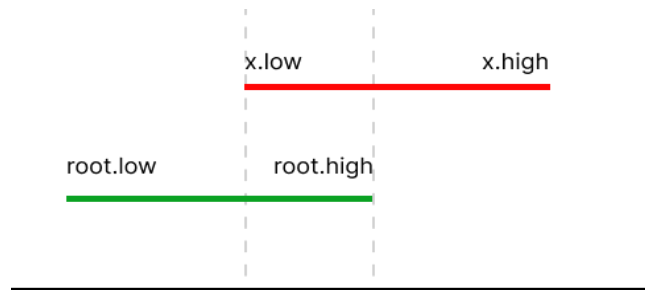


Рисунок 5 – Случай 1

2. Интервал  $x$  частично перекрывает интервал  $root$ , при этом  $x$  не выходит за границы  $root$  (рисунок 6).

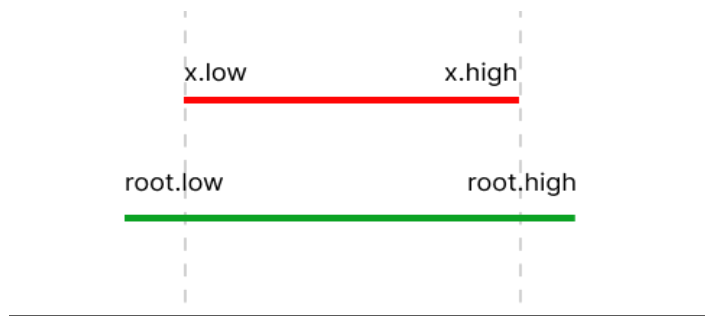


Рисунок 6 – Случай 2

3. Интервал  $x$  полностью перекрывает интервал  $root$ , при этом  $x$  выходит границы  $root$  (рисунок 7).

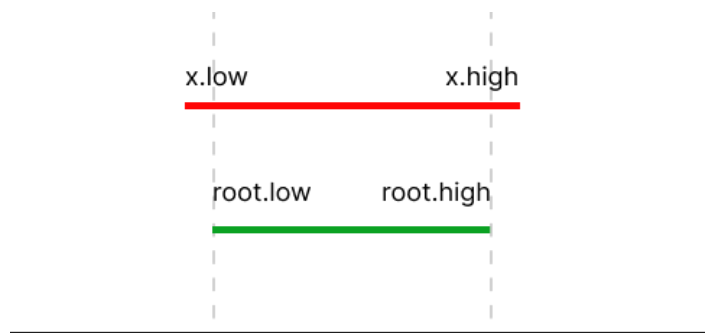


Рисунок 7 – Случай 3

4. Интервал  $x$  частично перекрывает интервал  $root$ , при этом  $x$  выходит за левую границу интервала  $root$  (рисунок 8).

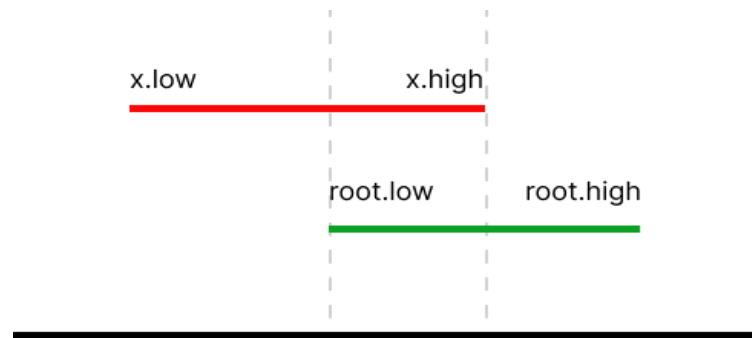


Рисунок 8 – Случай 4

5. Интервал  $x$  не перекрывает интервал  $root$ , при этом  $x$  лежит правее интервала  $root$  (рисунок 9).

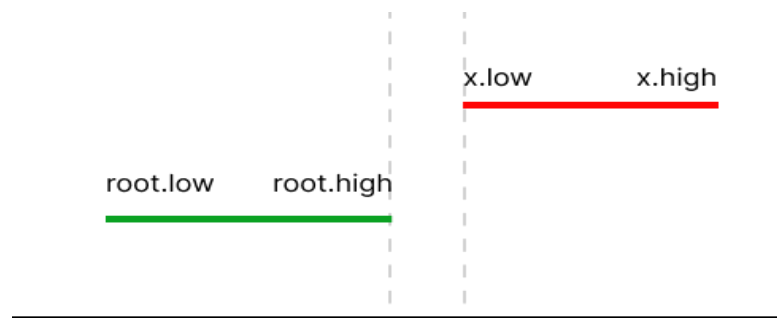


Рисунок 9 – Случай 5

6. Интервал  $x$  не перекрывает интервал  $root$ , при этом  $x$  лежит левее интервала  $root$  (рисунок 10).

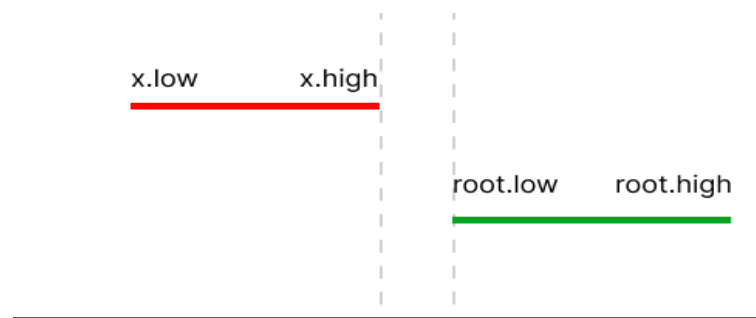


Рисунок 10 – Случай 6

### 4.3 Удаление узла

Для удаления узла рассматриваются две ситуации: первая ситуация – узел является листом, вторая ситуация – узел имеет хотя бы одного сына.

#### 1. Узел является листом.

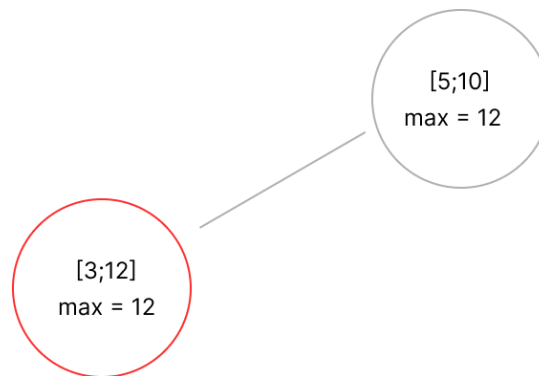


Рисунок 11 – Удаление узла

Удаляемый узел –  $[3; 12]$ ,  $\max = 12$ . Если у узла нет обоих сыновей, то узел удаляется и максимум в родительском пересчитывается. Таким образом в дереве (рисунок 11) остается только один узел -  $[5; 10]$ ,  $\max = 10$  (рисунок 12).

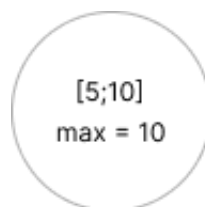


Рисунок 12 – Результат удаления

#### 2. Узел не является листом

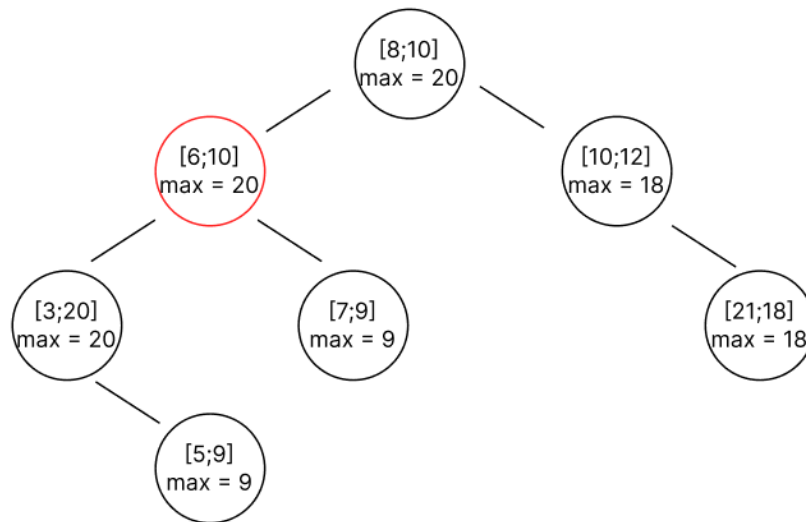


Рисунок 13 – Удаление узла [6;10], max = 20

Удаляемый узел – [6; 10], max = 20. У удаляемого узла есть сыновья, поэтому находится следующий узел по величине параметр *low* ([7; 9] max = 9). Затем в удаляемом узле интервал - [6; 10] меняет на [7; 9] и узел [7; 9] max = 9 удаляется, так как у него нет сыновей.

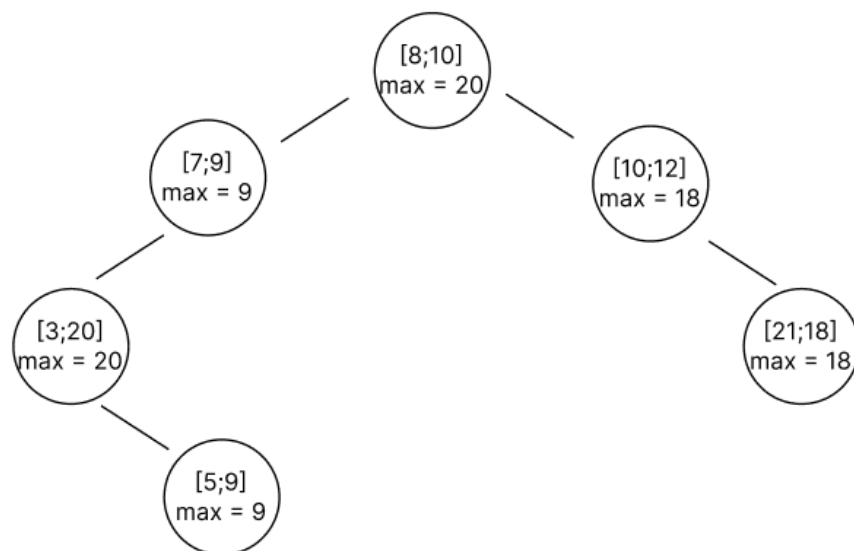


Рисунок 14 – Результат удаления

## **5 Формальная постановка задачи**

- 1) Изучить структуру данных “Дерево интервалов”;
- 2) Реализовать для дерева интервалов следующие операции:
  1. Добавление
  2. Удаление
  3. Поиск перекрываемых интервалов
- 3) Выполнить исследование на производительность (время работы);
- 4) Реализовать систему автоматического тестирования;
- 5) Результаты проделанной работы выложить в среде GitHub:
  1. Отчет
  2. Презентация
  3. Программный код
  4. Тесты
  5. Пакет CATS

## 6 Реализация

### 6.1 Добавление

*root.Insert(x)*, где *root* – дерево, *x* – добавляемый интервал

```
1. public Node Insert(Interval x)
2.     {
3.         if (this == null)
4.             return new Node(x, x.high);
5.         else
6.             {
7.                 this._Insert(this, x);
8.                 return this;
9.             }
10.    }
11.    Node _Insert(Node root, Interval x)
12.    {
13.        if (root.range.Equals(x))
14.        {
15.            return null;
16.        }
17.        else
18.        {
19.            if (root.max < x.high)
20.                root.max = x.high;
21.            if (root.range.low >= x.low)
22.            {
23.                if (root.left == null)
24.                {
25.                    root.left = new Node(x, x.high);
26.                    return root;
27.                }
28.                else
29.                    this._Insert(root.left, x);
30.            }
31.            else
32.            {
33.                if (root.right == null)
34.                {
35.                    root.right = new Node(x, x.high);
36.                    return root;
37.                }
38.                else
39.                    this._Insert(root.right, x);
40.            }
41.            return root;
42.        }
43.    }
```



## 6.2 Поиск перекрывающихся интервалов

*root.Insert(x)*, где *root* – дерево, *x* – интервал

```
1. public string Search(Interval x)
2.     {
3.
4.         if (this.range.low >= x.low && this.range.high
5.             <= x.high)
6.         {
7.             msg += $"{x} полностью перекрывает
8.             {this.range}\n";
9.
10.        }
11.        else if (this.range.low > x.high ||
12.                this.range.high < x.low)
13.        {
14.            msg += $"{x} перекрывает {this.range}\n";
15.        }
16.        else
17.        {
18.            msg += $"{x} не перекрывает {this.range}\n"
19.            ;
20.        }
21.        if (this.left != null)
22.        {
23.            this.left.Search(x);
24.        }
25.        if (this.right != null)
26.        {
27.            this.right.Search(x);
28.        }
29.        return msg;
30.    }
31. }
```

## 6.3 Удаление

*root.Remove(x)*, где *root* – дерево, *x* –удаляемый интервал

```
1. public Node Delete(Interval x)
2.     {
3.         if (root == null)
4.         {
5.             Console.WriteLine("Список интервалов пуст");
6.             return root;
7.         }
8.         if (root.range.low == x.low && root.range.high == x.high)
9.         {
10.            if (root.left == null && root.right == null)
11.            {
12.                root.range.low = 0;
13.                root.range.high = 0;
14.                root.max = 0;
15.
16.                return root;
17.            }
18.            if (root.left != null || root.right != null)
19.            {
20.                root_replace(root);
21.                return root;
22.            }
23.
24.            Console.WriteLine("Узел в дереве");
25.            return root;
26.        }
27.        if (root.range.low >= x.low && root.left != null)
28.        {
29.            remove(root.left, x);
30.        }
31.
32.        else if (root.range.low <= x.low && root.right
33.        != null)
34.        {
35.            remove(root.right, x);
36.        }
37.        else
38.            Console.WriteLine("Такого узла нет в дереве");
39.    }
```

## 7 Тестирование

Набор тестов состоит из 37 тестов на корректность работы алгоритма и 90 тестов на производительность.

Тест на корректность находится в отдельной папке, которая имеет следующий вид:

Тесты (папка)

1 – папка с тестом под номером 1

1.in.txt

1.out.txt

1.ans.txt

:

37 – папка с тестом под номером 37

37.in.txt

37.out.txt

37.ans.txt

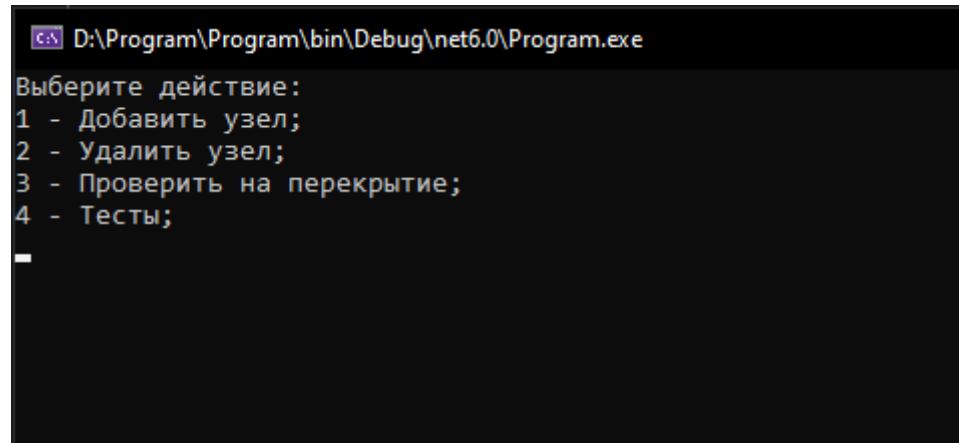
Файл “in” содержит входные данные. Тип операции, которая проверяется в тесте определяется по первой строке:

- 1) “insert”, то будет выполняться вставка узлов в дерево;
- 2) “search”, то будет выполняться проверка на перекрытие;
- 3) “delete”, то будет выполняться удаление узла из дерева.

Тесты на производительность создаются с помощью программного кода, где узлы генерируются случайным образом. И выводят на экран количество узлов в тесте и время, за которое выполнилась операция.

Запуск тестов происходит через меню, в котором при нажатии на кнопку “4” сначала запускаются тесты на корректность, а затем запускаются тесты на производительность.

Пример запуска тестов:

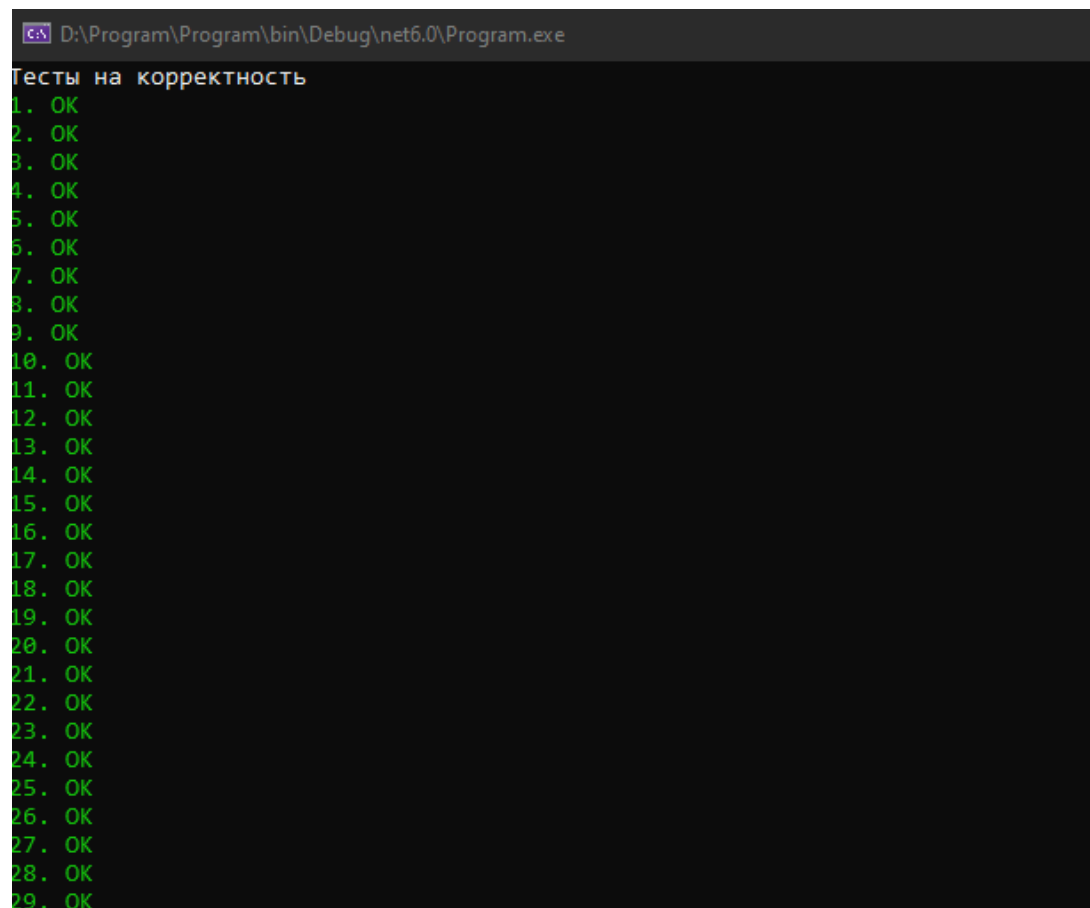


```

c:\> D:\Program\Program\bin\Debug\net6.0\Program.exe
Выберите действие:
1 - Добавить узел;
2 - Удалить узел;
3 - Проверить на перекрытие;
4 - Тесты;

```

Рисунок 18. Меню



```

c:\> D:\Program\Program\bin\Debug\net6.0\Program.exe
Тесты на корректность
1. OK
2. OK
3. OK
4. OK
5. OK
6. OK
7. OK
8. OK
9. OK
10. OK
11. OK
12. OK
13. OK
14. OK
15. OK
16. OK
17. OK
18. OK
19. OK
20. OK
21. OK
22. OK
23. OK
24. OK
25. OK
26. OK
27. OK
28. OK
29. OK

```

Рисунок 19. Тесты на корректность

```
cs D:\Program\Program\bin\Debug\net6.0\Pro
Тесты на производительность
100 0,121
200 0,21
300 0,3689
400 0,552
500 0,7566
600 0,8947
700 1,0786
800 1,1609
900 2,2676
1000 1,6243
1100 1,5971
1200 2,209
1300 3,207
1400 2,4887
1500 2,5616
1600 2,8796
1700 3,018
1800 3,5345
1900 3,2073
2000 3,2146
2100 3,681
2200 4,8276
2300 4,4785
2400 4,8728
2500 4,3142
2600 5,91
2700 4,5528
2800 4,7863
2900 5,4765
```

Рисунок 20. Тесты на производительность

## 8 Исследование

Исследование времени работы алгоритма состоит из трех частей, где первая часть – это анализ времени работы добавления узла в дерево, вторая часть – анализ времени работы удаления узла из дерева, третья часть – анализ поиска перекрываемых интервалов.

### 8.1 Добавления узла в дерево

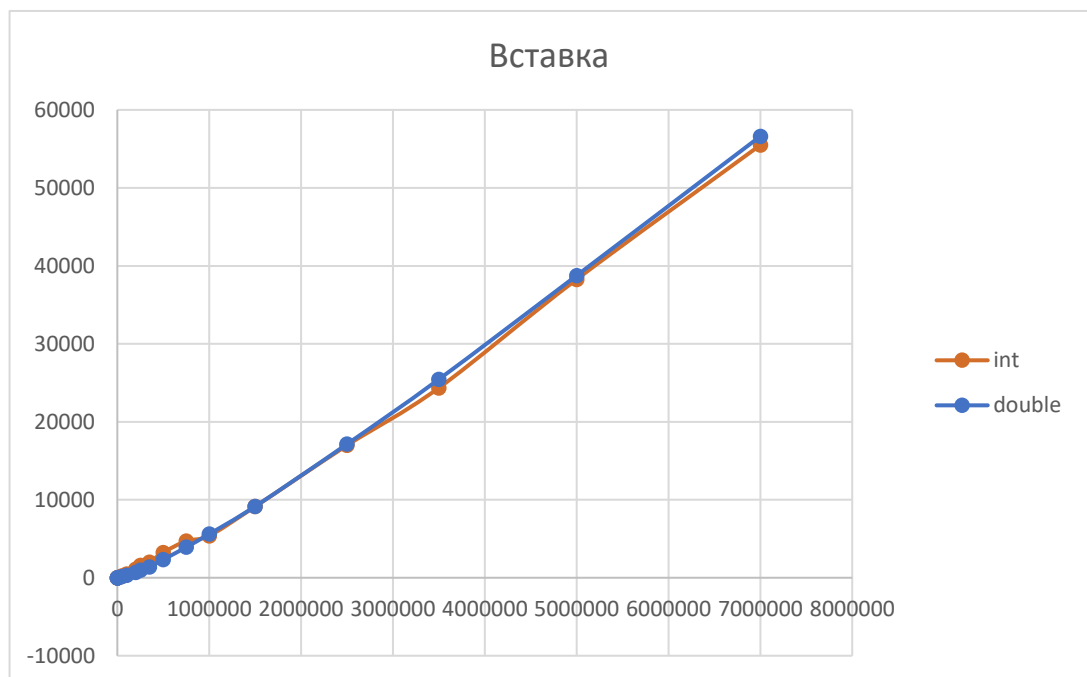


Рисунок 21. Время работы добавления

## 8.2 Анализ удаления узла из дерева



Рисунок 22. Время работы удаления

## 8.3 Анализ удаления узла из дерева

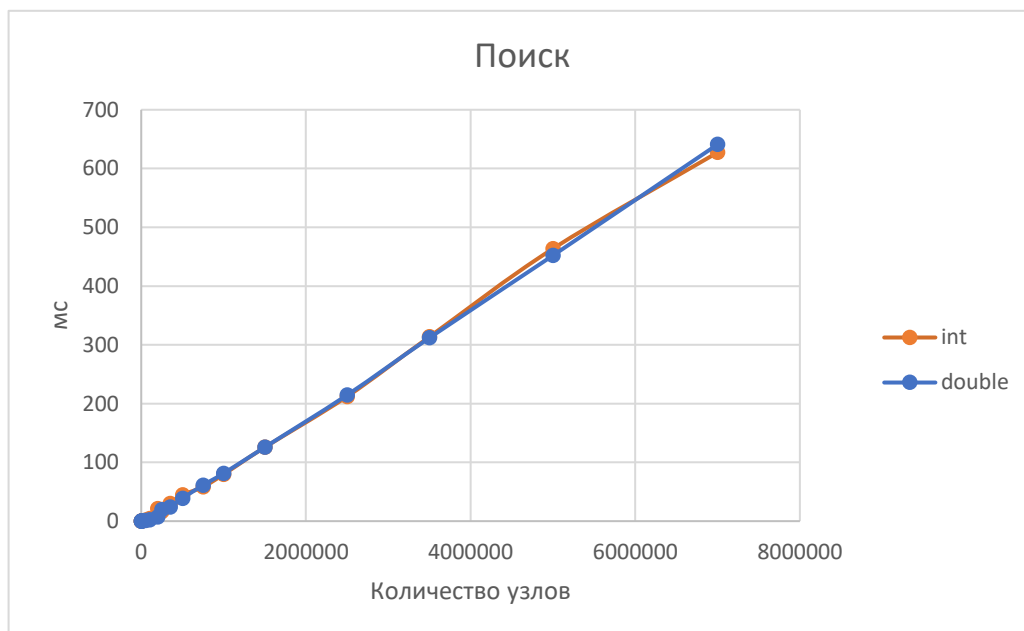


Рисунок 23. Время работы проверки на перекрытие

## 8.4 Вывод

	int	double	int	double	int	double
	вставка		удаление		поиск	
Кол-во узлов						
100	0,2161	0,0822	0,014	0,0052	0,0037	0,0022
500	1,3166	0,9349	0,0399	0,0179	0,0169	0,0086
1000	4,18	1,4663	0,0653	0,0286	0,0348	0,0164
10000	43,71	31,8574	0,7579	0,2469	0,3503	0,2036
50000	249,41	143,5984	2,1386	1,4966	1,5206	1,179
100000	480,24	311,4224	5,7364	2,8179	4,2096	2,0597
200000	1099,75	704,6265	23,73	9,0006	21,2535	7,0522
250000	1583,48	970,9418	21,1109	20,5778	16,2862	19,6583
350000	2022,53	1410,756	32,8571	28,7789	30,0864	24,1261
500000	3247,776	2336,368	49,3543	43,0751	44,5112	38,8277
750000	4713,675	3916,609	67,4035	69,0067	58,1684	61,1629
1000000	5384,338	5608,239	91,4292	89,3802	79,5766	81,3239
1500000	9130,773	9157,174	139,8762	140,8983	125,6112	125,8056
2500000	16975,59	17141,9	243,1205	243,3735	212,0627	214,5535
3500000	24353,17	25450,09	350,1961	351,9049	313,6168	311,8677
5000000	38268,49	38759,56	505,8355	507,5541	463,5499	452,0043
7000000	55495,1	56622,49	721,8165	728,2858	627,3314	641,0152

Время выполнения операций зависит от количества узлов в дереве. Чем больше узлов в дереве, тем дольше выполняется операция. Так же чем больше цифр после запятой, тем дольше выполняется операция.

Время работы операций не зависит от того какого типа данные содержатся в дереве (int или double).



## Список литературы

### Электронные ресурсы

1. Дерево интервалов (interval tree) и пересечение точки с множеством интервалов [Электронный ресурс] // neerc.ifmo. — Режим доступа: [Дерево интервалов \(interval tree\) и пересечение точки с множеством интервалов — Викиконспекты \(ifmo.ru\)](#)
2. Interval Tree. [Электронный ресурс] // geeksforgeeks. — Режим доступа: [Interval Tree - GeeksforGeeks](#)
3. Interval Trees: One step beyond BST [Электронный ресурс] // iq.opengenus. — Режим доступа: [Interval Trees: One step beyond BST \(opengenus.org\)](#)
4. Interval tree [Электронный ресурс] // wikipedia. — Режим доступа: [Interval tree - Wikipedia](#)
5. node-interval-tree [Электронный ресурс] // npm. — Режим доступа: [node-interval-tree - npm \(npmjs.com\)](#)
6. 5script/interval – three [Электронный ресурс] // github. — Режим доступа: [GitHub - 5script/interval-tree: A C++ header only interval tree implementation.](#)
7. INTERVAL TREES [Электронный ресурс] // dgp.toronto. — Режим доступа: [CSC378: Interval Trees \(toronto.edu\)](#)
8. Interval tree [Электронный ресурс] // cmu.edu. — Режим доступа: [intervaltrees.pdf \(cmu.edu\)](#)
9. Windowing queries [Электронный ресурс] // personal.us.es. — Режим доступа: [Lecture 8: Windowing queries \(us.es\)](#)
10. Interval tree [Электронный ресурс] // tutorialandexample. — Режим доступа: [Interval Tree - TAE \(tutorialandexample.com\)](#)
11. Interval tree [Электронный ресурс] // formulasearchengine. — Режим доступа: [Interval tree - formulasearchengine](#)

12. Interval tree [Электронный ресурс] // TutorialCup. — Режим доступа: [Interval Tree - Interval Tree in Data Structure Interval Tree \(tutorialcup.com\)](https://www.tutorialcup.com/Interval-Tree-in-Data-Structure-Interval-Tree/)
13. Динамические структуры данных [Электронный ресурс] // штегше. — Режим доступа: [Динамические структуры данных - интуит](#)
14. Дерево диапазонов [Электронный ресурс] // wikibrief. — Режим доступа: [Дерево диапазонов - Range tree \(wikibrief.org\)](https://wikibrief.org/Range-tree/)
15. Segment Tree Range Minimum Query [Электронный ресурс] // youtube. — Режим доступа: [Segment Tree Range Minimum Query - YouTube](#)
16. Interval tree [Электронный ресурс] // homepages.math. — Режим доступа: [Interval Trees \(uic.edu\)](https://homepages.math.uic.edu/~bayerh/IntervalTrees/)
17. Дерево Интервалов (Отрезков) [Электронный ресурс] // coolsoftware. — Режим доступа: [Дерево Интервалов \(Отрезков\) | Cool Software Blog](#)
18. Interval tree [Электронный ресурс] // drdobbs. — Режим доступа: [Interval Trees | Dr Dobb's \(drdobbs.com\)](https://drdobbs.com/Interval-Trees/)
19. Data Structures: Augmented Interval Tree to search for intervals overlapping [Электронный ресурс] // davismol. — Режим доступа: [Data Structures: Augmented Interval Tree to search for intervals overlapping | Dede Blog \(davismol.net\)](#)
20. Augmented Interval Tree in C# [Электронный ресурс] // Software Salad. — Режим доступа: [Augmented Interval Tree in C# | Software Salad \(wordpress.com\)](#)
21. Дерево (структура данных) [Электронный ресурс] // wikipedia. — Режим доступа: [Дерево \(структура данных\) — Википедия \(wikipedia.org\)](#)
22. Методы [Электронный ресурс] // microsoft. — Режим доступа: [Методы. Руководство по программированию на C# | Microsoft Learn](#)
23. Коллекция [Электронный ресурс] // wikipedia. — Режим доступа: [Коллекция \(программирование\) — Википедия \(wikipedia.org\)](#)

24. Интервал [Электронный ресурс] // wikipedia. — Режим доступа: [Интервал — Википедия \(wikipedia.org\)](https://ru.wikipedia.org/wiki/Интервал)
25. Оценка сложности алгоритмов [Электронный ресурс] // tproger. — Режим доступа: [Оценка сложности алгоритмов, или Что такое  \$O\(\log n\)\$  \(tproger.ru\)](https://tproger.ru/оценка-сложности-алгоритмов-или-что-такое-o-log-n/)
26. Временная сложность алгоритма [Электронный ресурс] // wikipedia. — Режим доступа: [Временная сложность алгоритма — Википедия \(wikipedia.org\)](https://ru.wikipedia.org/wiki/Временная_сложность_алгоритма)
27. Типы данных [Электронный ресурс] // wikipedia. — Режим доступа: [Типы данных в C# —метанит\(metanit.com\)](https://metanit.com/типы-данных-в-c/)