

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. Б9121-09.03.03пикд  Чурганов Никита Сергеевич  \_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Доклад защищен:  С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | *(подпись)*  Руководитель  Доцент ИМКТ А.С Кленин  *(должность, уч. звание)*  \_\_\_\_\_\_\_\_\_\_\_\_\_  *(подпись)*  «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2022г. |
| Рег. № \_\_\_\_\_\_  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г. |  |  |

г. Владивосток

2022г

**Оглавление**

[Глоссарий 3](#_Toc125562888)

[Введение 4](#_Toc125562889)

[1 История развития 6](#_Toc125562890)

[2 Суть алгоритма 7](#_Toc125562891)

[3 Построение дерева интервалов 8](#_Toc125562892)

[4 Операции над деревом 10](#_Toc125562893)

[4.1 Добавление 10](#_Toc125562894)

[4.2 Проверка на перекрытие 10](#_Toc125562895)

[4.3 Удаление узла 13](#_Toc125562896)

[5 Формальная постановка задачи 15](#_Toc125562897)

[6 Реализация 16](#_Toc125562898)

[6.1 Добавление 16](#_Toc125562899)

[6.2 Поиск перекрываемых интервалов 17](#_Toc125562900)

[6.3 Удаление 18](#_Toc125562901)

[7 Тестирование 19](#_Toc125562902)

[8 Исследование 22](#_Toc125562903)

[8.1 Добавления узла в дерево 22](#_Toc125562904)

[8.2 Анализ удаления узла из дерева 23](#_Toc125562905)

[8.3 Анализ удаления узла из дерева 24](#_Toc125562906)

[8.4 Вывод 25](#_Toc125562907)

[**Список литературы** 26](#_Toc125562908)

## 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]

**Кореневой** **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

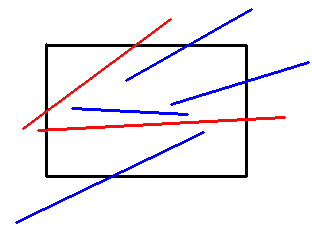
Интервальное дерево – это древовидная структура данных, узел которой представляет интервал и максимальное значение в поддереве, поэтому характеризуется начальным и конечным значениями. В частности, дерево интервалов позволяет эффективно находить все интервалы, которые перекрываются любым заданным интервалом (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 и множество запросов. Каждый запрос характеризуется интервалом . Для каждого запроса необходимо определить множество отрезков из , которые перекрывают .

Построение дерева интервалов занимает время , а также памяти. На каждый запрос дерево интервалов позволяет отвечать за , где – 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 , так как , то переходим к левому сыну (рисунок 2).

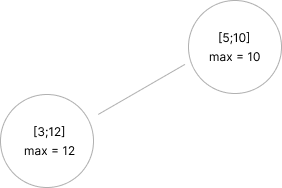


Рисунок 2 – Добавление узла в дерево

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).

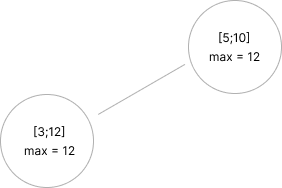


Рисунок 3 – Вычисление максимума в дереве

Аналогично предыдущим действиям добавляются другие узлы

(рисунок 4).

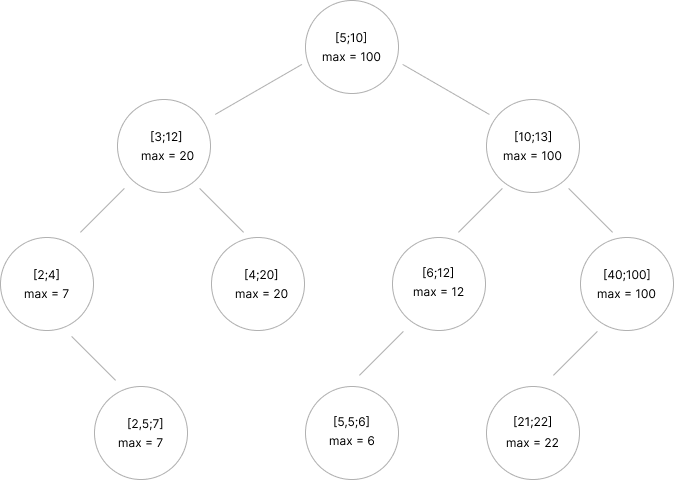


Рисунок 4 – Полученное дерево

## 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 *х*. В ходе прямого обхода выполняется сравнение границ интервала текущего узла дерева и границ интервала *х*. Для корректного выполнения операции перекрытия дерево должно иметь хотя бы один узел.

Введём обозначения:

– нижняя граница интервала ,

– верхняя граница интервала ;

– нижняя граница интервала ,

– верхняя граница интервала .

Варианты перекрытия интервалов:

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

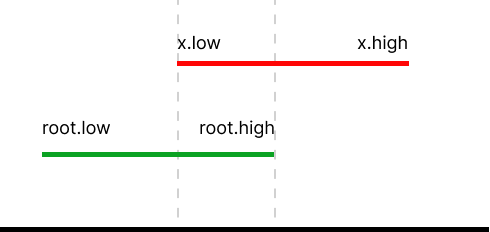


Рисунок 5 – Случай 1

1. Интервал частично перекрывает интервал , при этом не выходит заграницы (рисунок 6).

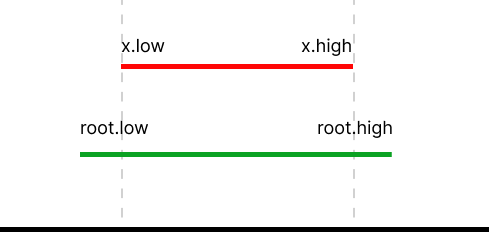


Рисунок 6 – Случай 2

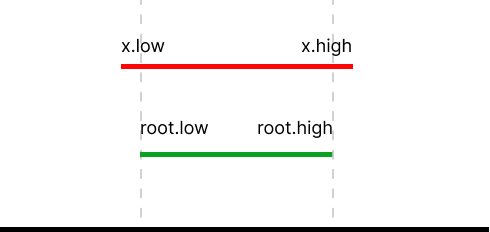
1. Интервал полностью перекрывает интервал , при этом выходит границы (рисунок 7).

Рисунок 7 – Случай 3

1. Интервал частично перекрывает интервал , при этом выходит за левую границу интервала (рисунок 8).



Рисунок 8 – Случай 4

1. Интервал не перекрывает интервал , при этом лежит правее интервала (рисунок 9).

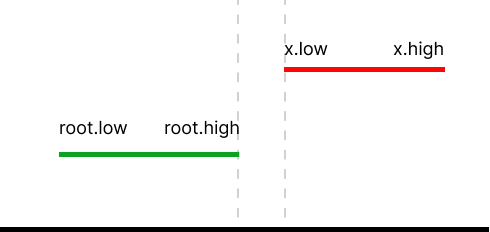


Рисунок 9 – Случай5

1. Интервал не перекрывает интервал , при этом лежит левее интервала (рисунок 10).

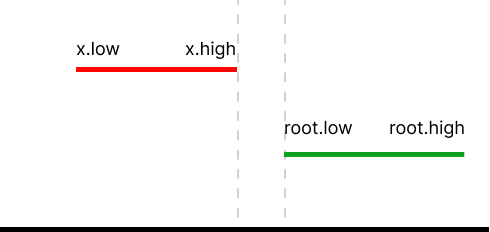


Рисунок 10 – Случай 6

## 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.

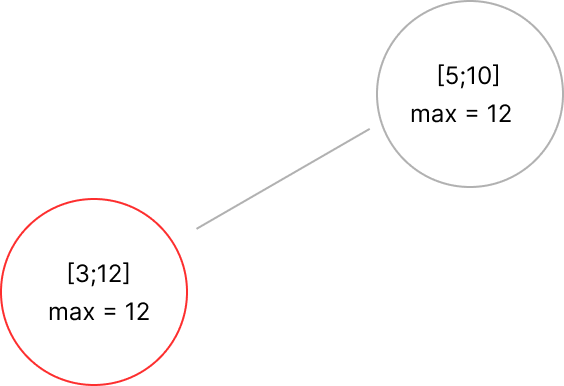


Рисунок 11 – Удаление узла

Удаляемый узел – . Если у узла нет обоих сыновей, то узел удаляется и максимум в родительском пересчитывается. Таким образом в дереве (рисунок 11) остается только один узел - (рисунок 12).



Рисунок 12 – Результат удаления

1. Узел не является листом

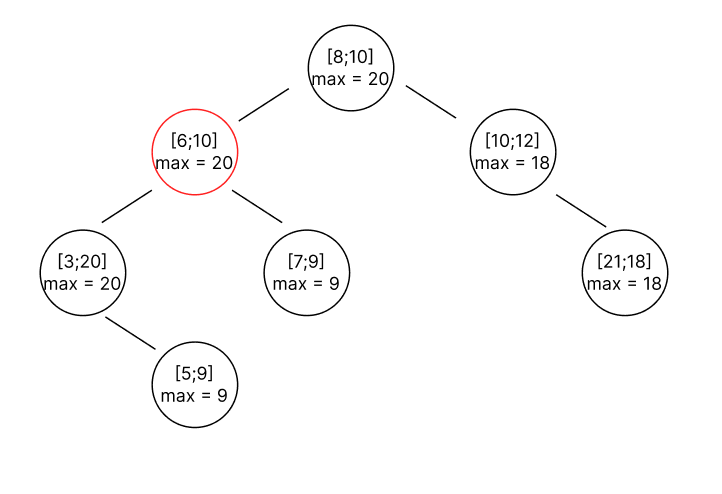


Рисунок 13 – Удаление узла [6;10], max = 20

Удаляемый узел – . У удаляемого узла есть сыновья, поэтому находится слудующий узел по величине параметр (). Затем в удаляемом узле интервал - меняет на и узел удаляется, так как у него нет сыновей.

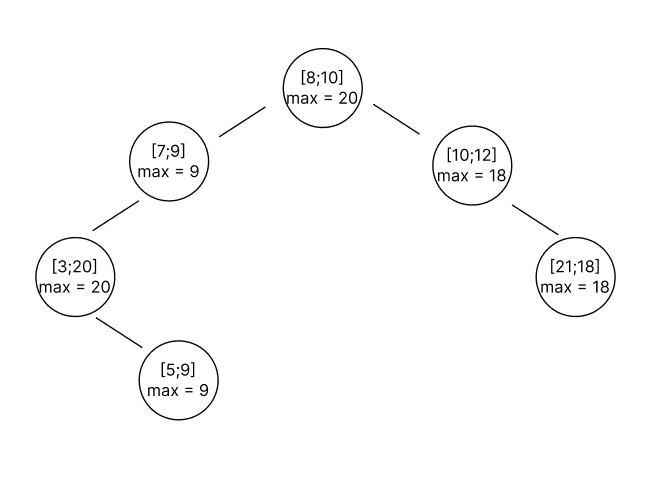


Рисунок 14 – Результат удаления

## 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

, где – дерево, – 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 перекрываемых интервалов

, где – дерево, – интервал

1. public string Search(Interval x)
2. {
4. if (this.range.low >= x.low && this.range.high

<= x.high)

1. {
2. msg += $"{x} полностью перекрывает

{this.range}\n";

3. }
4. else if (this.range.low > x.high ||

this.range.high < x.low)

1. {
2. msg += $"{x}  перекрывает {this.range}\n";
4. }
5. else
6. {
7. msg += $"{x} не перекрывает {this.range}\n";
9. }
11. if (this.left != null)
12. {
13. this.left.Search(x);
14. }
15. if (this.right != null)
16. {
17. this.right.Search(x);
18. }
19. return msg;
21. }

## 6.3 Удаление

, где – дерево, –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;
16. **return** root;
17. }
18. **if** (root.left != **null** || root.right != **null**)
19. {
20. **root\_replace(root);**
21. **return** root;
22. }
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. **}**

**else** **if** (root.range.low <= x.low && root.right != **null**)

1. {
2. **remove**(root.right, x);
3. }
4. **else**
5. Console.WriteLine("Такого узла нет в дереве");
6. }

## 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.

Изображение выглядит как текст

Автоматически созданное описаниеПример запуска тестов:

Рисунок 18. Меню

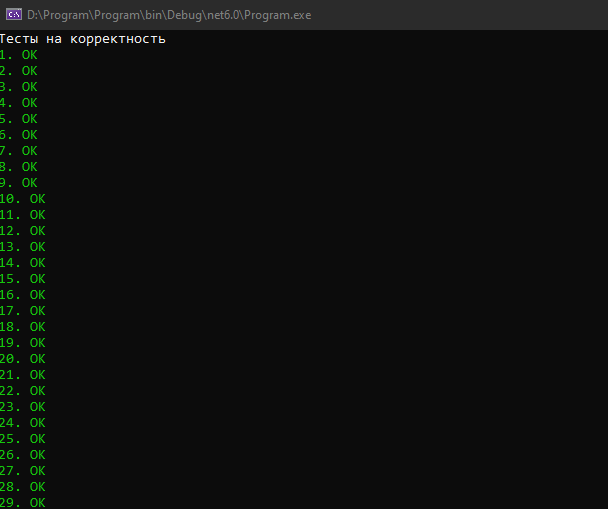


Рисунок 19. Тесты на корректность

Изображение выглядит как текст

Автоматически созданное описание

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

|  |  |  |
| --- | --- | --- |
| Вставка | | |
| Количество узлов | Время вставки всех узлов в миллисекундах | Время вставки одного узла в миллисекундах |
| 100 | 0,121 | 0,00121 |
| 500 | 0,21 | 0,00042 |
| 1000 | 1,46 | 0,00146 |
| 2000 | 3,2146 | 0,001607 |
| 5000 | 9,5803 | 0,001916 |
| 10000 | 20,7423 | 0,002074 |
| 50000 | 149,047 | 0,002981 |
| 100000 | 283 | 0,00283 |
| 350000 | 1370,86 | 0,003917 |
| 600000 | 2912 | 0,004853 |
| 1000000 | 5398 | 0,005398 |
| 2000000 | 12284 | 0,006142 |

## 8.2 Анализ удаления узла из дерева

Рисунок 22. Время работы удаления

|  |  |
| --- | --- |
| Количество узлов в дереве | Время удаления в миллисекундах |
| 100 | 0,0121 |
| 500 | 0,023 |
| 1000 | 0,054 |
| 2000 | 0,0847 |
| 5000 | 0,343 |
| 10000 | 0,688 |
| 50000 | 5,56 |
| 100000 | 9,28 |
| 350000 | 64 |
| 600000 | 170 |
| 1000000 | 201 |
| 2000000 | 260 |

## 8.3 Анализ удаления узла из дерева

Рисунок 23. Время работы проверки на перекрытие

|  |  |
| --- | --- |
| Количество узлов в дереве | Время проверки в миллисекундах |
| 100 | 0,0056 |
| 500 | 0,015 |
| 1000 | 0,029 |
| 2000 | 0,05 |
| 5000 | 0,3 |
| 10000 | 1 |
| 50000 | 3 |
| 100000 | 8,6 |
| 350000 | 33 |
| 600000 | 48 |
| 1000000 | 93 |
| 2000000 | 188 |

## 8.4 Вывод

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

# 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)](https://neerc.ifmo.ru/wiki/index.php?title=%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D0%B2%D0%B0%D0%BB%D0%BE%D0%B2_(interval_tree)_%D0%B8_%D0%BF%D0%B5%D1%80%D0%B5%D1%81%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5_%D1%82%D0%BE%D1%87%D0%BA%D0%B8_%D1%81_%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%BE%D0%BC_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D0%B2%D0%B0%D0%BB%D0%BE%D0%B2)
2. Interval Tree. [Электронный ресурс] // geeksforgeeks. — Режим доступа: [Interval Tree - GeeksforGeeks](https://www.geeksforgeeks.org/interval-tree/)
3. Interval Trees: One step beyond BST [Электронный ресурс] // iq.opengenus. — Режим доступа: [Interval Trees: One step beyond BST (opengenus.org)](https://iq.opengenus.org/interval-tree/)
4. Interval tree [Электронный ресурс] // wikipedia. — Режим доступа: [Interval tree - Wikipedia](https://en.wikipedia.org/wiki/Interval_tree#:~:text=%D0%92%20%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B5%20%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE%20%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D0%B2%D0%B0%D0%BB%D0%BE%D0%B2%2D%D1%8D%D1%82%D0%BE%20%D0%B4%D1%80%D0%B5%D0%B2%D0%BE%D0%B2%D0%B8%D0%B4%D0%BD%D0%B0%D1%8F,%D0%B2%D0%B8%D0%B4%D0%B8%D0%BC%D1%8B%D1%85%20%D1%8D%D0%BB%D0%B5%D0%BC%D0%B5%D0%BD%D1%82%D0%BE%D0%B2%20%D0%B2%D0%BD%D1%83%D1%82%D1%80%D0%B8%20%D1%82%D1%80%D0%B5%D1%85%D0%BC%D0%B5%D1%80%D0%BD%D0%BE%D0%B9%20%D1%81%D1%86%D0%B5%D0%BD%D1%8B)
5. node-interval-tree [Электронный ресурс] // npm. — Режим доступа: [node-interval-tree - npm (npmjs.com)](https://www.npmjs.com/package/node-interval-tree)
6. 5cript/interval – three [Электронный ресурс] // github. — Режим доступа: [GitHub - 5cript/interval-tree: A C++ header only interval tree implementation.](https://github.com/5cript/interval-tree)
7. INTERVAL TREES [Электронный ресурс] // dgp.toronto. — Режим доступа: [CSC378: Interval Trees (toronto.edu)](https://www.dgp.toronto.edu/public_user/JamesStewart/378notes/22intervals/)
8. Interval tree [Электронный ресурс] // cmu.edu. — Режим доступа: [intervaltrees.pdf (cmu.edu)](https://www.cs.cmu.edu/~ckingsf/bioinfo-lectures/intervaltrees.pdf)
9. Windowing queries [Электронный ресурс] // personal.us.es. — Режим доступа: [Lecture 8: Windowing queries (us.es)](https://personal.us.es/almar/cg/08windowing.pdf)
10. Interval tree [Электронный ресурс] // tutorialandexample. — Режим доступа: [Interval Tree - TAE (tutorialandexample.com)](https://www.tutorialandexample.com/interval-tree)
11. Interval tree [Электронный ресурс] // formulasearchengine. — Режим доступа: [Interval tree - formulasearchengine](https://www.formulasearchengine.com/wiki/Interval_tree)
12. Interval tree [Электронный ресурс] // TutorialCup. — Режим доступа: [Interval Tree - Interval Tree in Data Structure Interval Tree (tutorialcup.com)](https://www.tutorialcup.com/interview/tree/interval-tree.htm)
13. Динамические структуры данных [Электронный ресурс] // штегше. — Режим доступа: [Динамические структуры данных - интуит](https://intuit.ru/studies/courses/648/504/lecture/11455#:~:text=%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5%20%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D1%8B%20%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85%20%E2%80%93%20%D1%8D%D1%82%D0%BE%20%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D1%8B,%D0%B8%20%D1%85%D0%B0%D1%80%D0%B0%D0%BA%D1%82%D0%B5%D1%80%20%D1%81%D0%B2%D1%8F%D0%B7%D0%B5%D0%B9%20%D0%BC%D0%B5%D0%B6%D0%B4%D1%83%20%D1%8D%D0%BB%D0%B5%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D0%BC%D0%B8.)
14. Дерево диапозонов [Электронный ресурс] // wikibrief. — Режим доступа: [Дерево диапазонов - Range tree (wikibrief.org)](https://ru.wikibrief.org/wiki/Range_tree#:~:text=%D0%92%20%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B5%2C%20%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE%20%D0%B4%D0%B8%D0%B0%D0%BF%D0%B0%D0%B7%D0%BE%D0%BD%D0%BE%D0%B2%20%2D,%D0%9B%D1%83%D0%B8%20%D0%91%D0%B5%D0%BD%D1%82%D0%BB%D0%B8%20%D0%B2%201979%20%D0%B3%D0%BE%D0%B4%D1%83)
15. Segment Tree Range Minimum Query [Электронный ресурс] // youtube. — Режим доступа: [Segment Tree Range Minimum Query - YouTube](https://www.youtube.com/watch?app=desktop&v=ZBHKZF5w4YU&t=61s)
16. Interval tree [Электронный ресурс] // homepages.math. — Режим доступа: [Interval Trees (uic.edu)](https://homepages.math.uic.edu/~jan/mcs481/windowing.pdf)
17. Дерево Интервалов (Отрезков) [Электронный ресурс] // coolsoftware. — Режим доступа: [Дерево Интервалов (Отрезков) | Cool Software Blog](https://blog.coolsoftware.ru/2014/01/blog-post/)
18. Interval tree [Электронный ресурс] // drdobbs. — Режим доступа: [Interval Trees | Dr Dobb's (drdobbs.com)](https://www.drdobbs.com/cpp/interval-trees/184401179)
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)](http://www.davismol.net/2016/02/07/data-structures-augmented-interval-tree-to-search-for-interval-overlapping/)
20. Augmented Interval Tree in C# [Электронный ресурс] // Software Salad. — Режим доступа: [Augmented Interval Tree in C# | Software Salad (wordpress.com)](https://brooknovak.wordpress.com/2013/12/07/augmented-interval-tree-in-c/)
21. Дерево (структура данных) [Электронный ресурс] // wikipedia. — Режим доступа: [Дерево (структура данных) — Википедия (wikipedia.org)](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_(%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85))
22. Методы [Электронный ресурс] // microsoft. — Режим доступа: [Методы. Руководство по программированию на C# | Microsoft Learn](https://learn.microsoft.com/ru-ru/dotnet/csharp/programming-guide/classes-and-structs/methods)
23. Коллекция [Электронный ресурс] // wikipedia. — Режим доступа: [Коллекция (программирование) — Википедия (wikipedia.org)](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BB%D0%BB%D0%B5%D0%BA%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5))
24. Интервал [Электронный ресурс] // wikipedia. — Режим доступа: [Интервал — Википедия (wikipedia.org)](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%B2%D0%B0%D0%BB)
25. Оценка сложности алгоритмов [Электронный ресурс] // tproger. — Режим доступа: [Оценка сложности алгоритмов, или Что такое О(log n) (tproger.ru)](https://tproger.ru/articles/computational-complexity-explained/)
26. Временная сложность алгоритма [Электронный ресурс] // wikipedia. — Режим доступа: [Временная сложность алгоритма — Википедия (wikipedia.org)](https://ru.wikipedia.org/wiki/%D0%92%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%B0)
27. Типы данных [Электронный ресурс] // wikipedia. — Режим доступа: [Типы данных в С# —метанит(metanit.com)](https://metanit.com/sharp/tutorial/2.1.php)