ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

*«*САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт машиностроения, материалов и транспорта

Высшая школа автоматизации и робототехники

КУРСОВАЯ РАБОТА

**Алгоритм «B – Дерево»**

по дисциплине «Объектно-ориентированное программирование»

|  |  |  |
| --- | --- | --- |
| Выполнил  Студент  гр. 3331506/90401 | *(подпись)* | Копейко И.В. |
| Работу принял | *(подпись)* | Ананьевский М.С. |

Санкт-Петербург

2022 г.

**Введение**

B-дерево (читается как Би-дерево) — это особый тип сбалансированного дерева поиска, в котором каждый узел может содержать более одного ключа и иметь более двух дочерних элементов. Из-за этого свойства B-дерево называют сильноветвящимся.

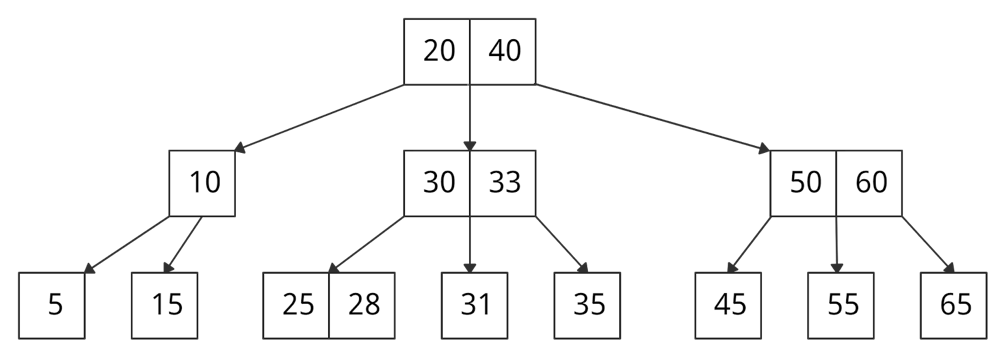


Рисунок 1 – Пример структуры B-дерева

Вторичные запоминающие устройства (жесткие диски, SSD) медленно работают с большим объемом данных. Людям захотелось сократить время доступа к физическим носителям информации, поэтому возникла потребность в таких структурах данных, которые способны это сделать. Помимо этого B-деревья используют:

* В базах данных и файловых системах.
* Для хранения блоков данных (вторичные носители).
* Для многоуровневой индексации.

Двоичное дерево поиска, АВЛ-дерево, красно-черное дерево и т. д. могут хранить только один ключ в одном узле. Если нужно хранить больше, высота деревьев резко начинает расти, из-за этого время доступа сильно увеличивается.

С B-деревом все не так. Оно позволяет хранить много ключей в одном узле и при этом может ссылаться на несколько дочерних узлов. Это значительно уменьшает высоту дерева и, соответственно, обеспечивает более быстрый доступ к диску.

**Описание алгоритма**

Дерево принимает только единственный параметр «t» или «T», который будет определять количество ключей и указателей в каждом узле. «t» определяет минимальное число указателей в узле. «T» является альтернативой, определяет максимальное число указателей. Иногда эти величины именуют «Б-фактор».

Имеются следующие правила:

1. В каждом узле содержатся минимум (t – 1) ключей и минимум (t) указателей. Все ключи и указатели расположен по возрастанию и чередуются между собой. Максимум ключей (2t – 1), а указателей (2t). Указателей всегда на 1 больше чем ключей.
2. Корень может иметь как минимум один ключ и два указателя, предел такой же как и у других узлов.
3. Потомок, на которого имеется указатель содержит ключи больше чем ключ слева от указателя и меньше чем ключ справа от указателя.
4. Листья потомков (указателей) не имеют.
5. Глубина (число уровней) всех ветвей всегда одинакова.
6. Новый ключ добавляется в самый нижний узел

Основные операция производимые с B-Деревом:

* Добавление ключа
* Поиск ключа
* Удаление ключа

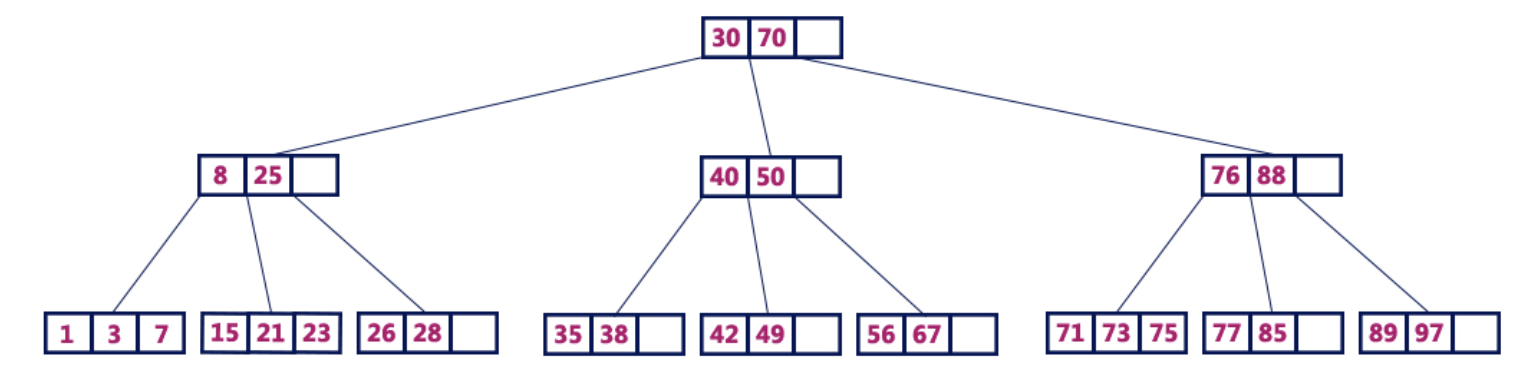
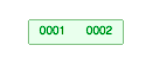


Рисунок 2 – B-Дерево с Т = 4

**Визуальная демонстрация алгоритма добавления**

*Пример сценария 1(обычное добавление):*



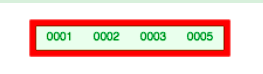
Добавляем 3



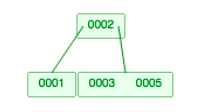
*Пример сценария 2 (деление коренного узла):*

**

Добавляем 5

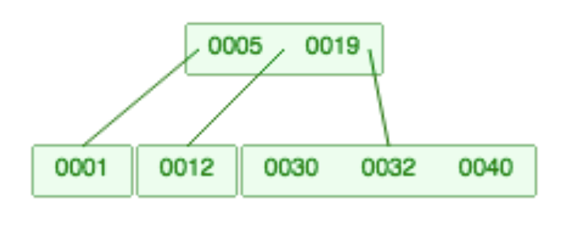


Узел переполнен, он делится

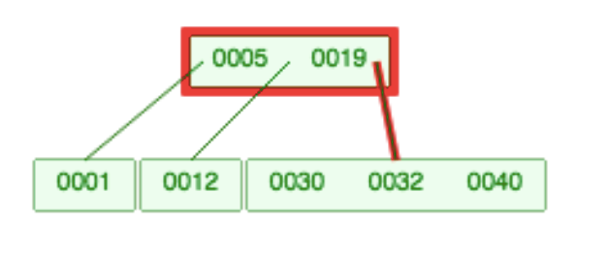


*Пример сценария 3 (деление узла):*

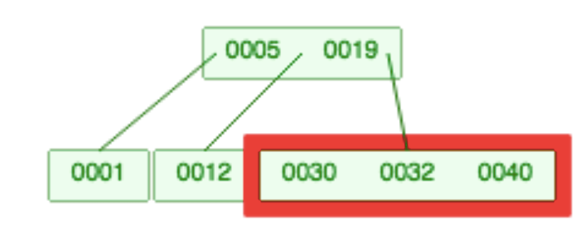
Имеется другое B-дерево с T = 4, решаем добавить ключ 25.



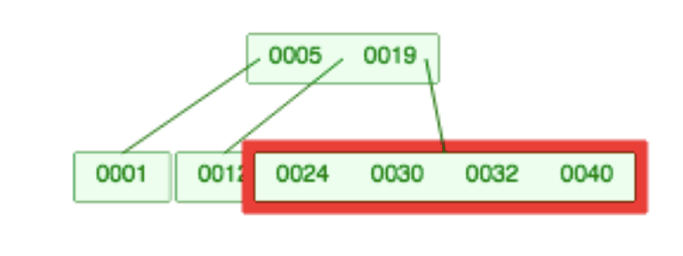
Мы переходим в правый узел



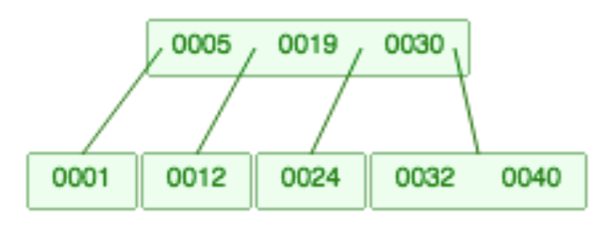
Добрались до нужного узла



Узел переполнен, его разбивают

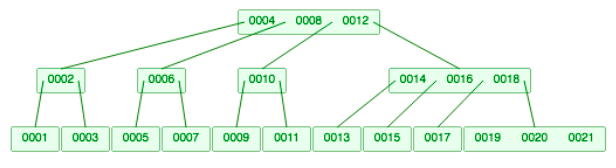


Итог операции добавления

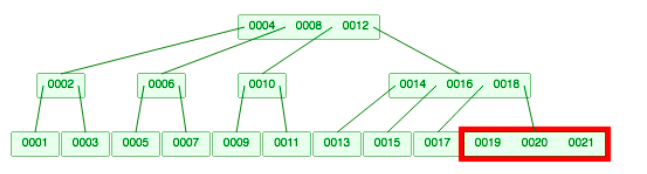


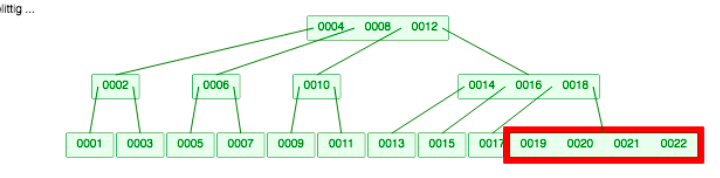
*Пример сценария 4 (деление узла рекурсивно):*

Имеется другое B-дерево с T = 4, решаем добавить ключ 22:

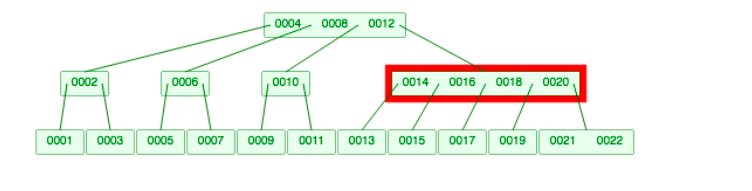


Добавляем 22 в нижний узел

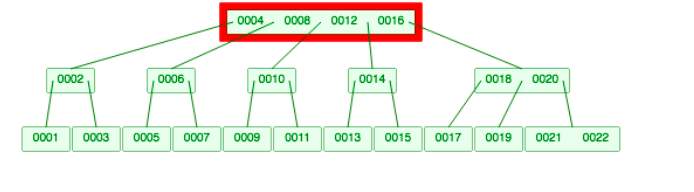




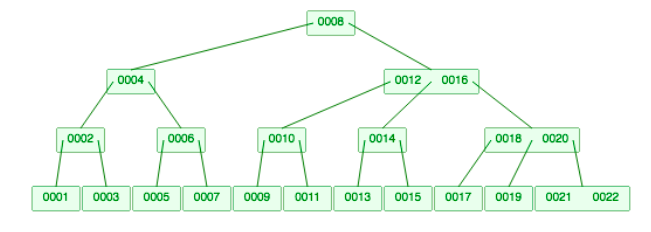
Узел переполнен, он делится



Один ключ отдали в узел уровнем выше, теперь он тоже переполнен и делится



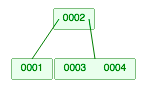
То же самое, делится коренной узел



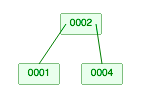
Дерево всегда растет вверх!

**Визуальная демонстрация алгоритма удаления**

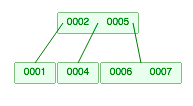
*Пример сценария 1 (простое удаление):*



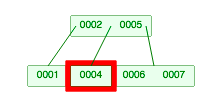
Удалим ключ 3



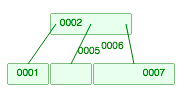
*Пример сценария 2 (взятие ключа у брата):*

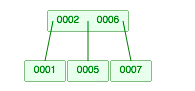


Удаляем ключ 4

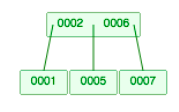


Он единственный в своем узле

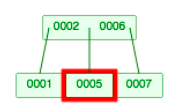


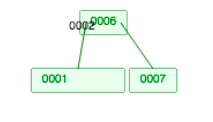


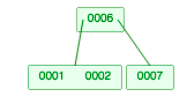
*Пример сценария 3 (объединяем узлы):*



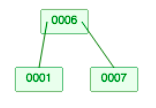
Удаляем ключ 5







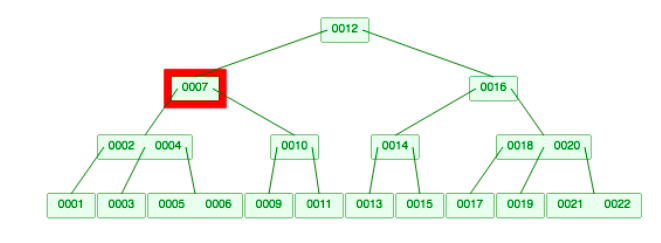
*Пример сценария 4 (объединяем коренной узел):*



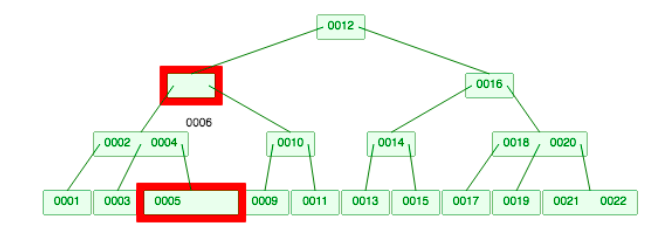
Удаляем ключ 1



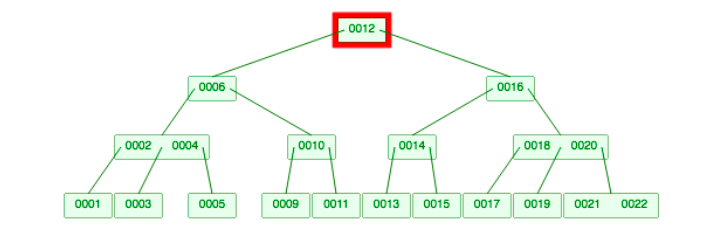
*Пример сценария 5 (берем ключ из нижнего узла):*

**

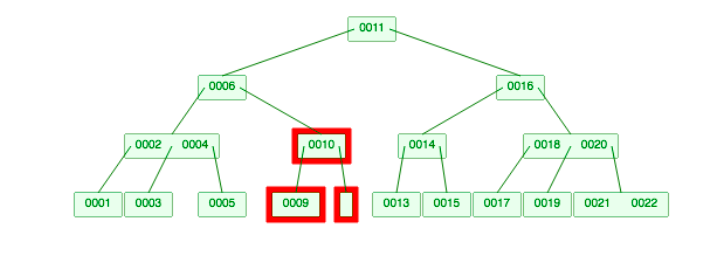
Удаляем ключ 7



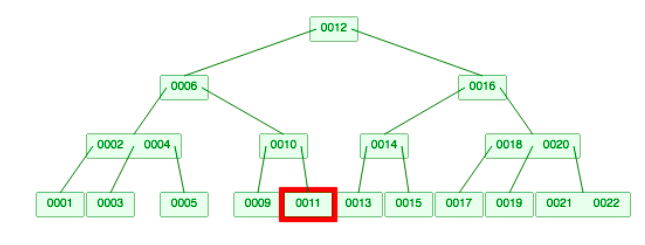
*Пример сценария 6 ( забираем ключи из нижнего узла):*



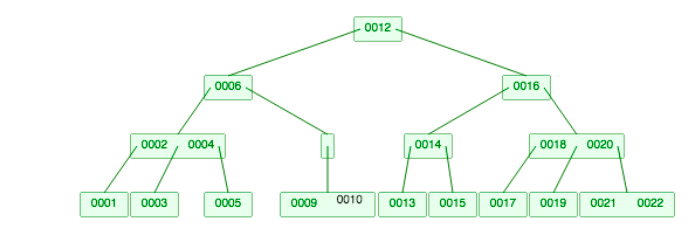
Удаляем ключ 12

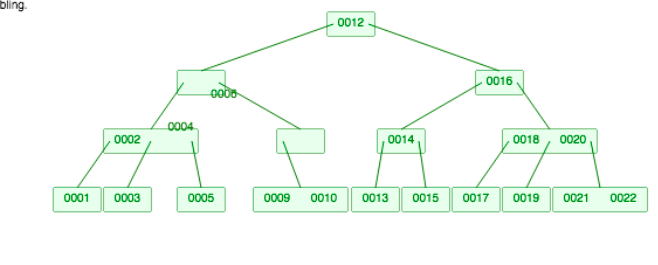


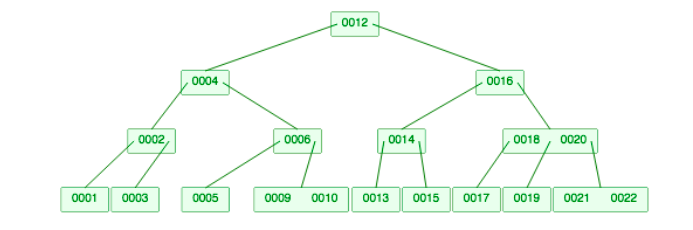
*Пример сценария 7 (рекурсивно забираем ключи):*

**

Удаляем ключ 11

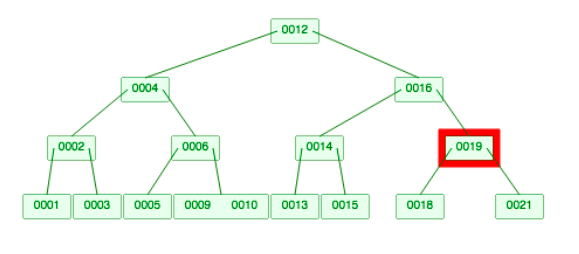




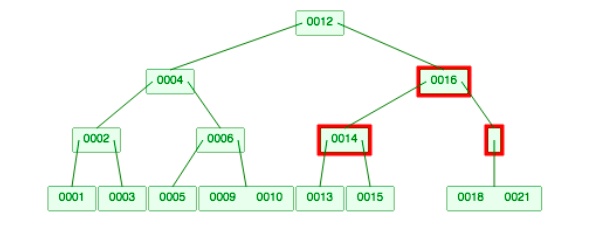


Вместе с ключом был передан еще и указатель

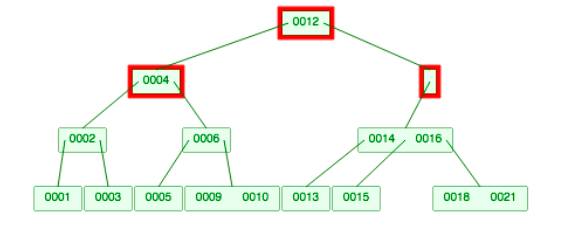
*Пример сценария 8 (рекурсивно забираем ключи):*

**

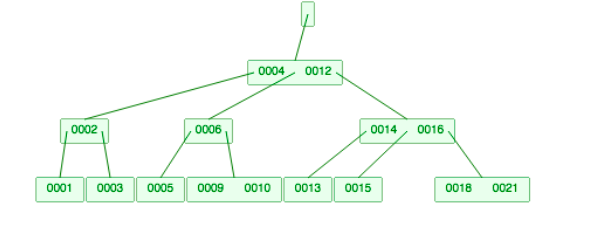
Удаляем ключ 19

**

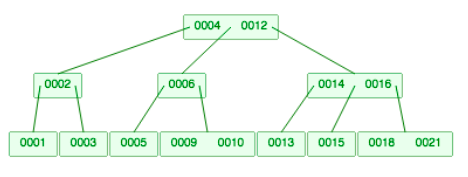
Нижние узлы под ключом 19 объединились, теперь будет происходить объединение с братом

**

Родителя больше не ключей, он тоже будет объединяться с братом

**

Так уменьшилась высота всего дерева



**Исследование алгоритма**

Исходя из интернет-источников временная сложность алгоритма для каждой из операций (вставка, поиск, удаление) равно O(log(n))

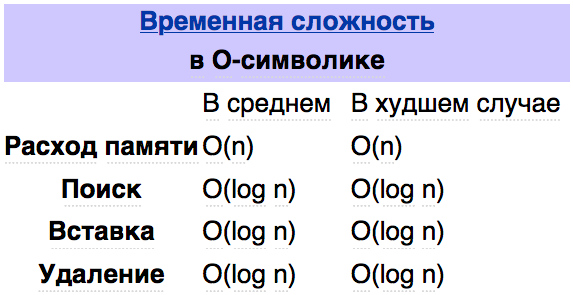


Рисунок 3 – Временная сложность алгоритма

Итак, исследуем временную сложность *алгоритма при добавлении*:

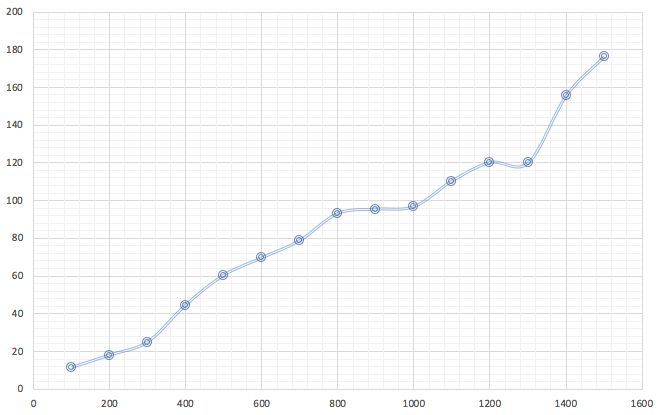


Рисунок 4 – Временная сложность алгоритма при добавлении на малом промежутке

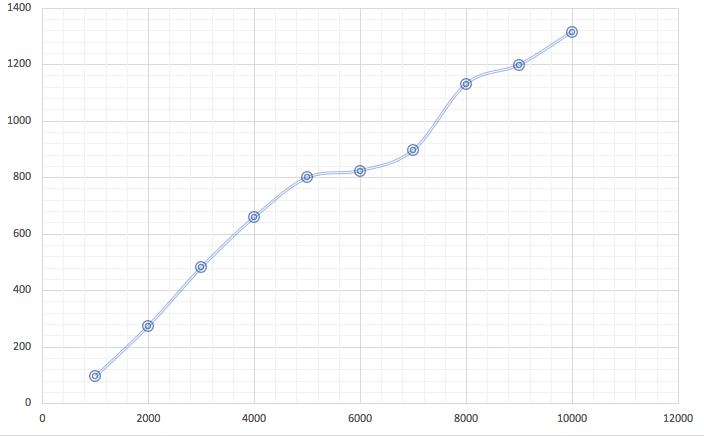
****

Рисунок 5 – Временная сложность алгоритма при добавлении на большом промежутке

На заявленный изначально график полученные результаты не похожи, скорее изменение происходит по линейному закону.

Исследуем временную сложность *алгоритма при удалении:*

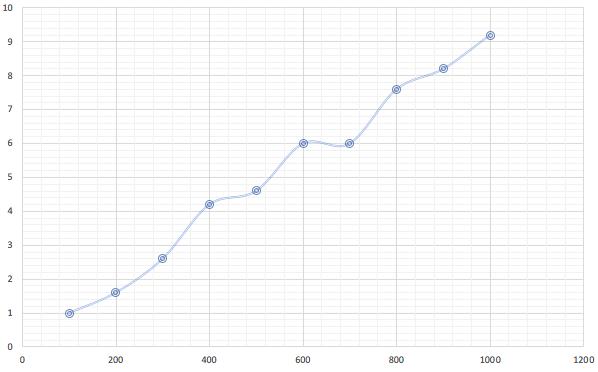


Рисунок 6 – Временная сложность алгоритма при удалении

Исследуем временную сложность *алгоритма при поиске:*

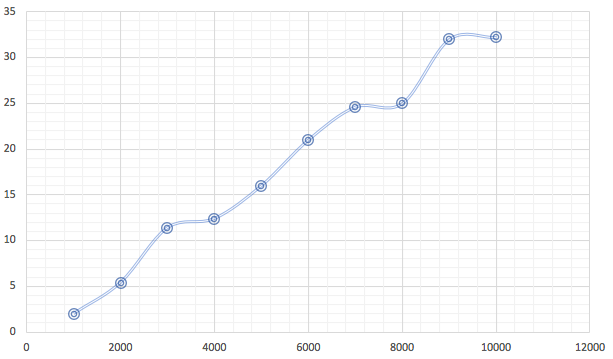


Рисунок 7 – Временная сложность алгоритма при поиске

Как видим все алгоритмы имеют линейную скачкообразную вертикальную функцию. Вероятно, идеальный график достигается на каком-то конкретном промежутке, или же алгоритм построения B-дерева в данной работе несколько отличается от эталона. Тем не менее, алгоритм поиска достаточно простой и должен быть схож с эталоном.

Все выполненные измерения проводились путем использования функции clock() из <ctime>.

**Заключение**

Алгоритм B – Дерево является довольно простым по своей сути, но имеет трудности на пути его реализации. Это касается того факта, что существуют разные сценарии поведения дерева при добавлении и удалении узла, которые необходимо прописать в коде, по этим причинам код становится довольно большим. На мой взгляд, должны существовать более простые алгоритмы для хранения данных.

**Список источников**

1. <https://youtu.be/WXXetwePSRk>
2. <https://youtu.be/GKa_t7fF8o0>
3. <https://www.cs.usfca.edu/~galles/visualization/BTree.html>
4. <https://codechick.io/tutorials/dsa/dsa-b-tree>
5. <https://habr.com/ru/post/114154/>
6. <http://cppstudio.com/post/468/>