**Министерство науки и высшего образования Российской Федерации**
**Федеральное государственное бюджетное образовательное учреждение высшего образования**
**«Российский химико-технологический университет имени Д.И. Менделеева»**
**Кафедра информационных компьютерных технологий**

# ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 7

Выполнил студент группы.....КС-33..... (Вагенлейтнер Никита Сергеевич)

Ссылка на репозиторий: .....(https://github.com/MUCTR-IKT-CPP/VagenlejtnerNS_33_alg.git)

Приняли: ..... Пысин Максим Дмитриевич
.....Краснов Дмитрий Олегович
.....Лобанов Алексей Владимирович
.....Крашенинников Роман Сергеевич

Дата сдачи: ..... (26.02.2025)

---

## Оглавление

## Описание задачи.

В рамках лабораторной работы необходимо изучить:

- Декартово дерево

Для этого его потребуется реализовать и сравнить в работе с реализованным ранее AVL-деревом.
Для анализа работы алгоритма понадобиться провести серии тестов:

- В одной серии тестов проводится 50 повторений
- Требуется провести серии тестов для N = 2^i элементов, при этом i от 10 до 18 включительно.

В рамках одной серии понадобится сделать следующее:

- Генерируем N случайных значений.
- Заполнить два дерева N количеством элементов в одинаковом порядке.
- Для каждого из серий тестов замерить максимальную глубину полученного деревьев.
- Для каждого дерева после заполнения провести 1000 операций вставки и замерить время.
- Для каждого дерева после заполнения провести 1000 операций удаления и замерить время.
- Для каждого дерева после заполнения провести 1000 операций поиска.
- Для каждого дерева замерить глубины всех веток дерева.

Для анализа структуры потребуется построить следующие графики:

- График зависимости среднего времени вставки от количества элементов в изначальном дереве для вашего варианта дерева и AVL дерева.
- График зависимости среднего времени удаления от количества элементов в изначальном дереве для вашего варианта дерева и AVL дерева.
- График зависимости среднего времени поиска от количества элементов в изначальном дереве для вашего варианта дерева и AVL дерева.
- График максимальной высоты полученного дерева в зависимости от N.
- Гистограмму среднего распределения максимальной высоты для последней серии тестов для AVL и для вашего варианта.
- Гистограмму среднего распределения высот веток в AVL дереве и для вашего варианта, для последней серии тестов.

Задания со звездочкой = + 5 дополнительных первичных баллов:

- Аналогичная серия тестов и сравнение ее для отсортированного заранее набора данных
- Реализовать красно черное дерево и провести все те же проверки с ним.

## Описание метода/модели.

### Декартово дерево

Декартово дерево (Treap) — это сочетание бинарного дерева поиска (BST) и кучи (Heap), которое поддерживает сбалансированную структуру без явной балансировки.

Свойства:

Бинарное дерево поиска (BST) по ключам:

- Для каждой вершины: ключи в левом поддереве меньше, а в правом — больше.

Двоичная куча (Min-Heap или Max-Heap) по приоритетам:

- Приоритет случайно выбирается при создании узла (обычно random()),
- У родителя приоритет выше, чем у потомков.

Операции:

Вставка:

- Выполняется как в BST (по ключу).
- Если нарушается свойство кучи, выполняется вращение (split/merge) для восстановления порядка.

Удаление:

- Найти элемент (обычный поиск в BST).
- Удалить, восстанавливая структуру слиянием поддеревьев.
- Split (разделение): Разделяет дерево на две части по заданному ключу.
- Merge (слияние): Объединяет два поддерева, сохраняя свойства Treap.

Сложность:

Операции (поиск, вставка, удаление) выполняются за O(log N) в среднем, так как случайные приоритеты обеспечивают хорошую балансировку.

Преимущества:

- Простая реализация без явной балансировки (в отличие от AVL).
- Хорошо работает на динамических множествах с операциями разбиения/объединения.
- Недостатки:
- Производительность зависит от качества генератора случайных чисел.
- Может быть менее эффективным, чем строго сбалансированные деревья (AVL, Red-Black).

## Выполнение задачи.

Для реализации и выполнения данного алгоритма был задействован язык Python. Были реализованы следующие программные блоки: алгоритмический блок, тестовый блок, аналитический блок полученных результатов работы алгоритма.

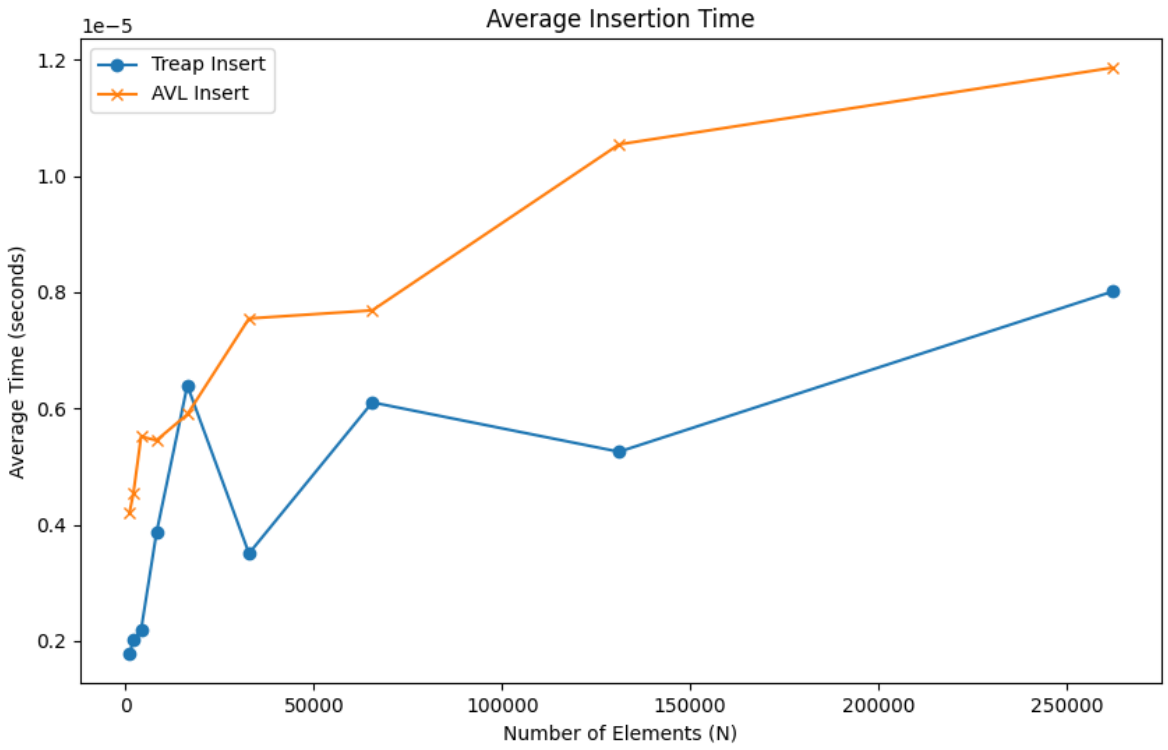
**Блок кода отвечающий за реализацию декартово дерева:**

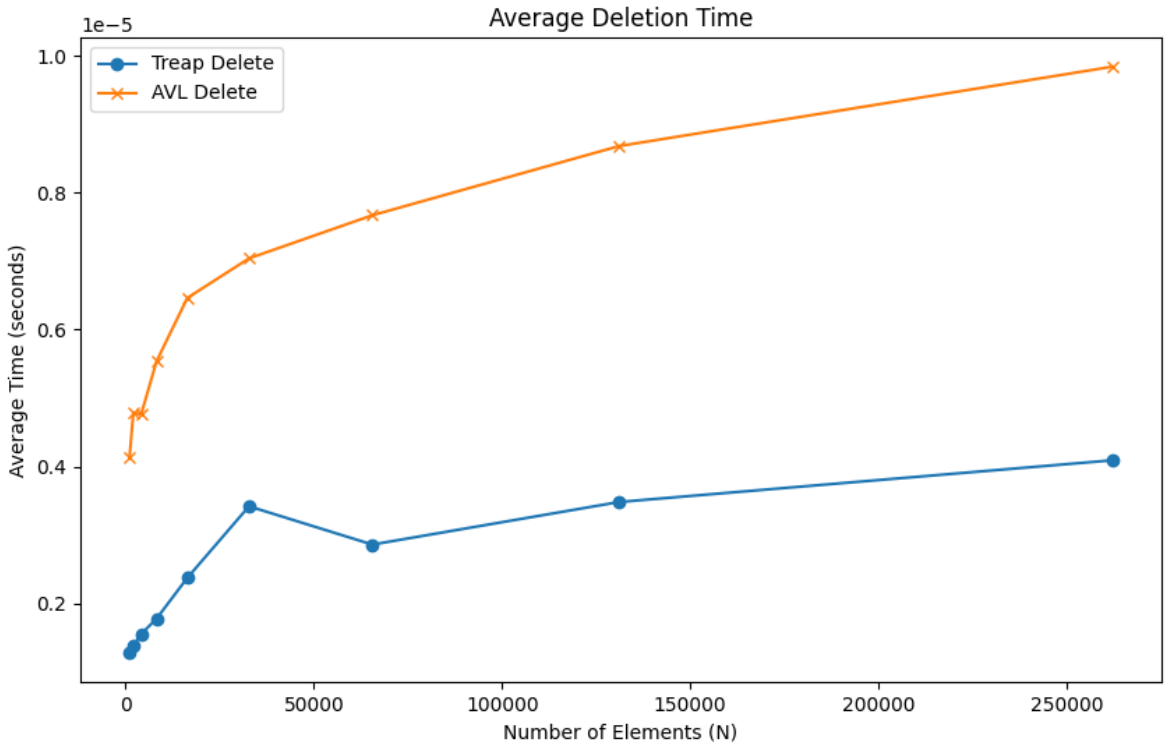
```
class TreapNode:
    def __init__(self, key):
        self.key = key
        self.priority = random.randint(1, 1000000)  # Случайный приоритет
        self.left = None
        self.right = None
class Treap:
    def __init__(self):
        self.root = None
    def _rotate_right(self, node):
        left = node.left
        node.left = left.right
        left.right = node
        return left
    def _rotate_left(self, node):
        right = node.right
        node.right = right.left
        right.left = node
        return right
    def _insert(self, node, key):
        if node is None:
            return TreapNode(key)
        if key < node.key:
            node.left = self._insert(node.left, key)
            if node.left.priority > node.priority:
                node = self._rotate_right(node)
        else:
            node.right = self._insert(node.right, key)
            if node.right.priority > node.priority:
                node = self._rotate_left(node)
        return node
    def insert(self, key):
        self.root = self._insert(self.root, key)
    def _delete(self, node, key):
        if node is None:
            return None
        if key < node.key:
            node.left = self._delete(node.left, key)
        elif key > node.key:
            node.right = self._delete(node.right, key)
```

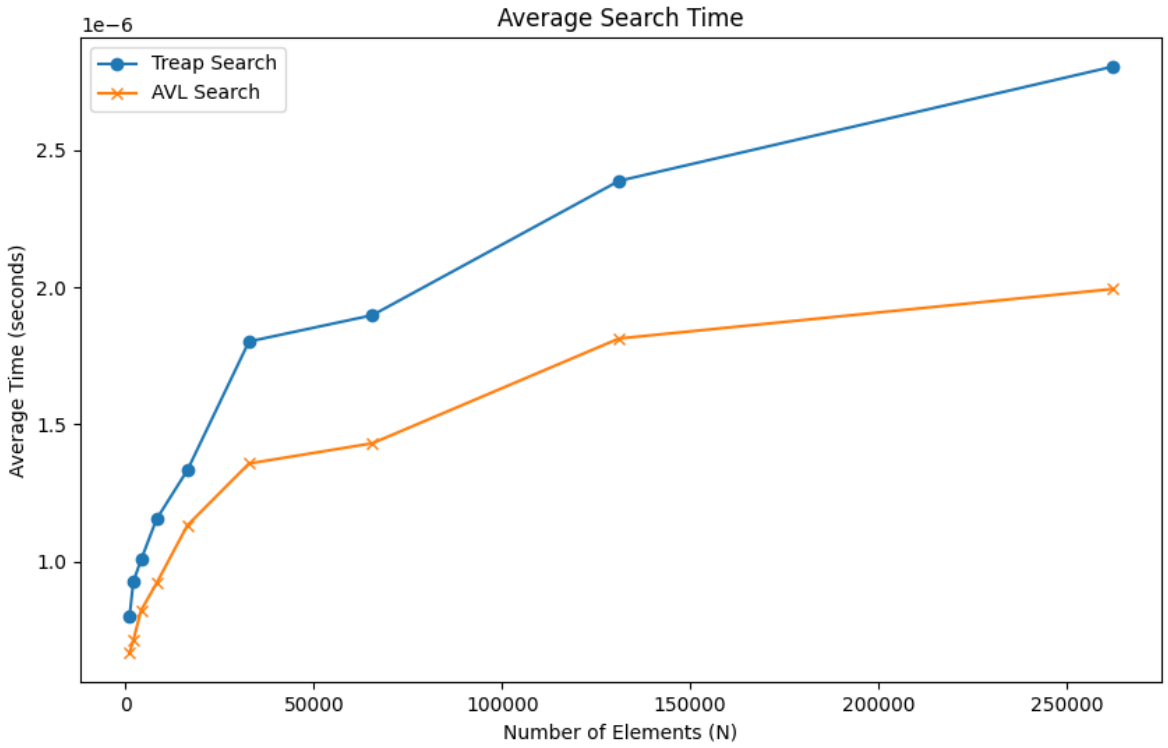
Блоки кода отвечающие за тестирование:

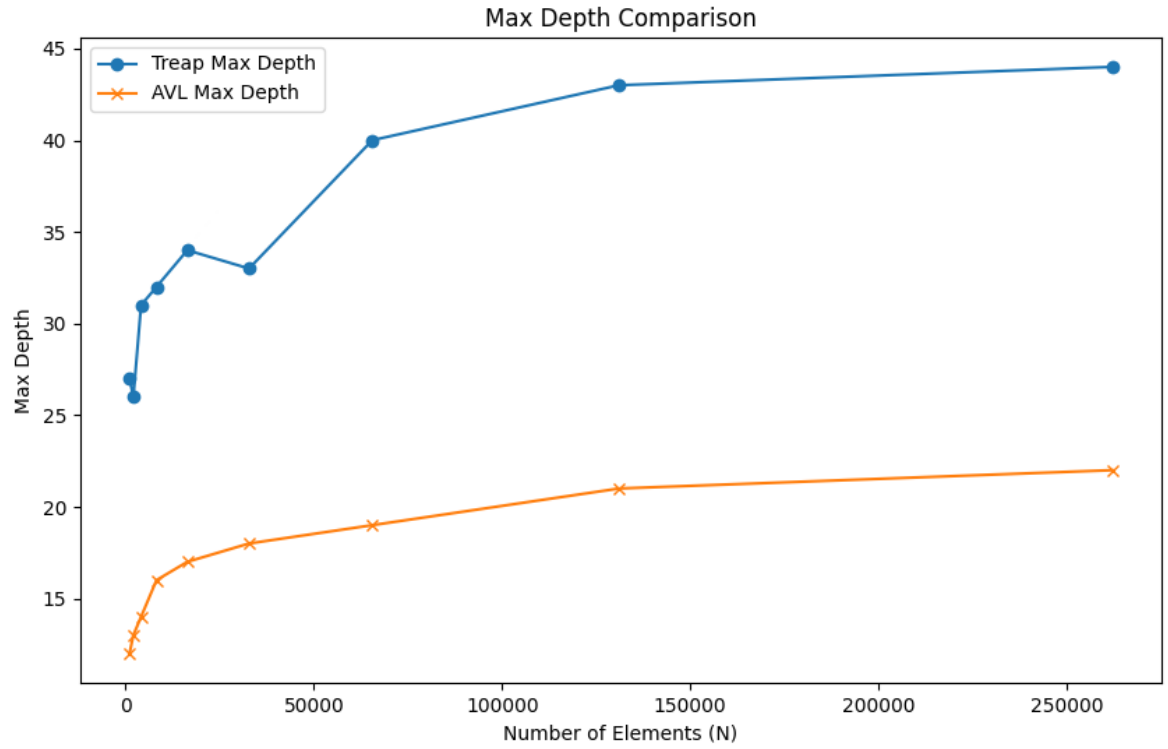
```
def run_tests():
    N_values = [2**i for i in range(10, 19)]
    treap_insert_times = []
    treap_delete_times = []
    treap_search_times = []
    avl_insert_times = []
    avl_delete_times = []
    avl_search_times = []
    treap_max_depths = []
    avl_max_depths = []
    for N in N_values:
        treap = Treap()
        avl = AVLTree()
        # Генерация случайных значений
        values = [random.randint(1, 1000000) for _ in range(N)]
        # Заполнение деревьев
        for value in values:
            treap.insert(value)
            avl.insert(value)
        # Измерение максимальной глубины
        treap_max_depths.append(treap.get_max_depth())
        avl_max_depths.append(avl.get_max_depth())
        # Вставка
        start_time = time.time()
        for value in values:
            treap.insert(value)
        treap_insert_times.append((time.time() - start_time) / N)
        start_time = time.time()
        for value in values:
            avl.insert(value)
        avl_insert_times.append((time.time() - start_time) / N)
        # Удаление
        start_time = time.time()
        for value in values:
            treap.delete(value)
        treap_delete_times.append((time.time() - start_time) / N)
        start_time = time.time()
        for value in values:
            avl.delete(value)
        avl_delete_times.append((time.time() - start_time) / N)
        # П
```

Результаты работы программы выводится в виде графика зависимости и приведены ниже.






## Заключение.

AVL-дерево гарантирует строгую балансировку и выполняет все операции за O(log N), но требует сложных поворотов при вставке и удалении.

Декартово дерево (Treap) использует случайные приоритеты для балансировки, достигая средней сложности O(log N) без строгого контроля высоты.

Сравнение производительности:

- AVL предпочтительно, если важна стабильность времени выполнения.
- Treap удобен для динамических структур, позволяя быстро разбивать и объединять деревья (операции split/merge).

Если требуются гарантированно сбалансированные деревья — AVL лучше. Если важны гибкость и простота реализации балансировки — Treap хороший выбор.