**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-36 Перминова П.А.

Приняли: Пысин Максим Дмитриевич

Краснов Дмитрий Олегович

Дата сдачи:

**Оглавление**

[**Описание задачи**](#_4ddts6wrv830) **3**

[**Описание метода/модели**](#_hau1q1wru7fi) **4**

[**Выполнение задачи**](#_h9jjnq6x7ohp) **7**

[**Заключение**](#_ewtnqcbebmrw) **10**

# Описание задачи

В рамках лабораторной работы необходимо изучить Рандомизированное дерево поиска.

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 измерить глубины всех веток дерева.

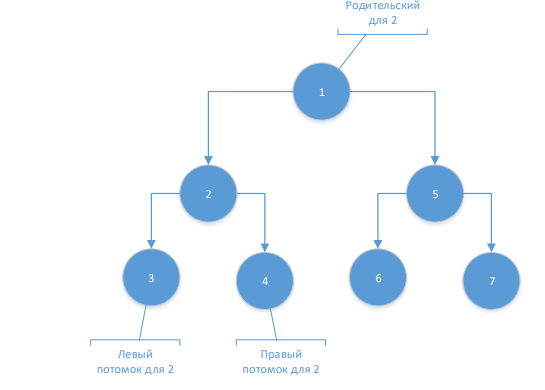
Для анализа структуры потребуется построить следующие графики:

* 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.

# Описание метода/модели

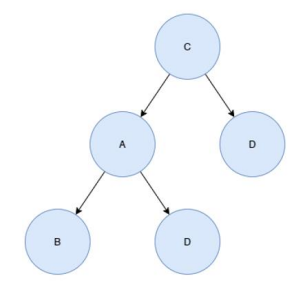
Двоичное дерево – это иерархическая структура, в которой каждый узел содержит не более чем двух потомков.

Для каждого узла, тот узел, который стоит выше по иерархии для него называют родительским узлом, а те узлы, что стоят ниже, для которых этот узел является родительским, называются правым и левым наследниками.



**AVL-дерево**

АВЛ-дерево - сбалансированное двоичное дерево поиска, в котором поддерживается следующее свойство: для каждой его вершины высота её двух поддеревьев различается не более чем на 1.



АВЛ-дерево с n ключами имеет высоту h = O(log n).

**Балансировка**

Балансировкой вершины называется операция, которая в случае разницы высот левого правого поддеревьев |h(L)−h(R)|=2, изменяет связи предок-потомок в поддереве данной вершины так, чтобы восстановилось свойство дерева |h(L)−h(R)|⩽1, иначе ничего не меняет. Для балансировки будем хранить для каждой вершины разницу между высотой её левого и правого поддерева diff[i]=h(L)−h(R)

Для балансировки вершины используются один из 4 типов вращений:

* Простой/малый левый поворот
* Простой/малый правый поворот
* Сложный/большой левый поворот – это два последовательных поворота, сначала правый потом левый.
* Сложный/большой правый поворот – это два последовательных поворота, сначала левый потом правый.

Все повороты выполняются относительно какого-либо узла.

Операция вставки и удаления вызываются практически так же, как у обычного двоичного дерева. Разница только в том, что для каждой вставки и каждого удаления требуется последним действием вызывать операцию перебалансировки, каждый раз проверяя от низа к верху необходимость ребалансировать дерево в текущем узле.

**Рандомизированное дерево**

Рандомизированное дерево это вариант дерева, которое хоть и не является

самобалансирующимся, но показывает хорошую асимптотическую сложность на больших количествах элементов.

Идея рандомизированного дерева исходит из того, что двоичное дерево само по себе является логарифмической структурой, если оно будет заполняться лучшим из возможных способов. Это подразумевает, что если достаточно хорошо перемещать данные перед тем как их укладывать в дерево, мы получим близкую к идеалу структуру, которая будет обладать сложностью поиска O(log(n)).

**Вставка**

В рамках классической вставки в дерево, мы ищем положение вставляемого элемента в дерево, и вставляем его так, чтобы оно оказалось в самом низу одной из его веток. В случае же рандомизированного дерева, вставка может осуществляется как стандартным путем, так и вставкой в корень, с определенно рассчитанной вероятностью.

**Вставка в корень**

В случае если бы наше дерево строилось идеально, то корнем этого дерева могла бы является каждая вершина с вероятностью 1 / (N + 1), где N количество вершин в принципе. Соответственно, нам следует выполнять вставку в корень именно с этой вероятностью отдельно взятой для каждой вершины в момент ее вставки в какое-либо поддерево.

**Удаление**

Удаление в случае с рандомизированным деревом можно реализовать двумя путями, либо классическим удалением из двоичного дерева. Либо удалением через слияние, в котором вместо приоритета корень полученного после слияния дерева будет выбираться случайным образом все так же в зависимости от размера поддеревьев.

# Выполнение задачи

Программа написана на языке программирования Python.

Классы, описанные в программе:

* Node (создание корневого узла, в котором есть конструктор, который содержит следующие поля: значение корневого узла, ссылки на левого и правого потомка, высоту узла и размер дерева.)
* Rand\_tree (класс, содержащий методы рандомизированного дерева, который содержит следующие методы: getsize(), fixsize(), find(), rotateR(), rotateL(), insertroot(), insert(), join(), remove(), height(), fixheight())
* AVL (класс, содержащий методы АВЛ-дерева. Он содержит следующие методы: height(), bfactor(), fixheight(), rotateR(), rotateL(), balance(), insert(), findmin(), find(), removemin() и remove().)

В программе создаются объекты классов Tree и AVL, выполняются девять тестов. В каждом тесте вычисляется размер списка, равный 2i, где i равно он 10 до 18 включительно. Далее выполняется заполнение деревьев числами из сгенерированного списка и по пятьдесят тестов для каждого из деревьев, в которых выполняется по 1000 операций вставки, поиска и удаления. Также выполняется расчет максимальной высоты дерева и высот всех его веток, замеряется время данных операция и в файл записывается среднее время каждой операции.

Ниже приведены графики зависимости времени от количества элементов массива для операций вставки, поиска и удаления. Использовалось среднее время выполнения тысячи операций.

График для операции вставки:

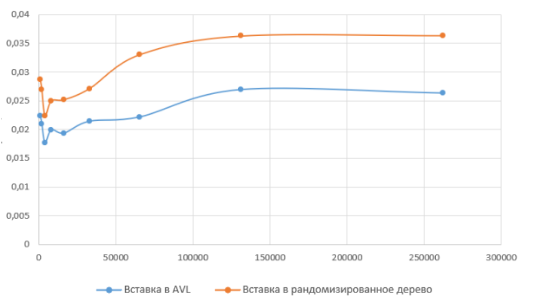


График для операции поиска:

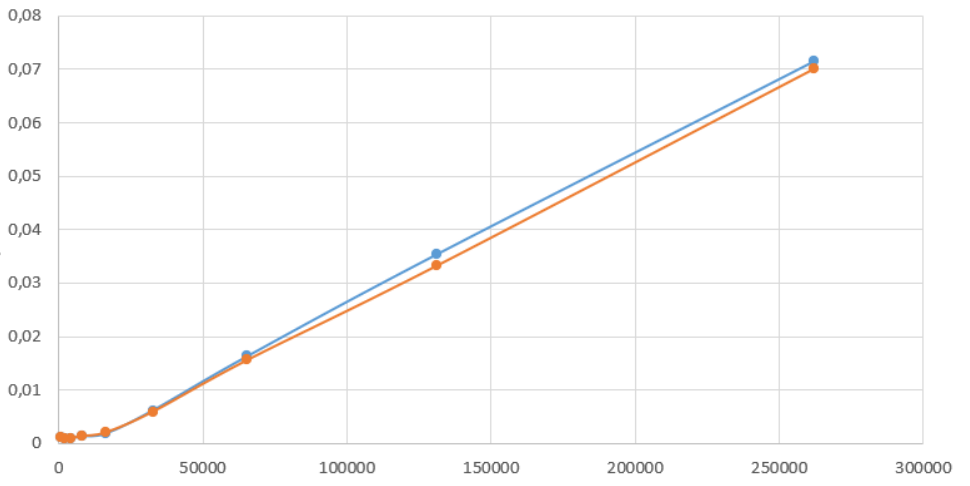
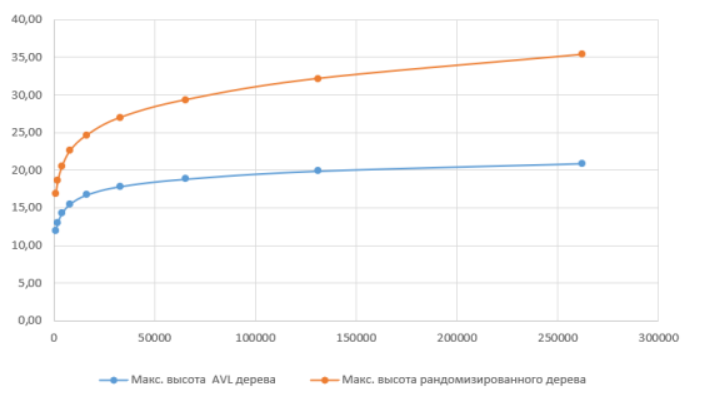


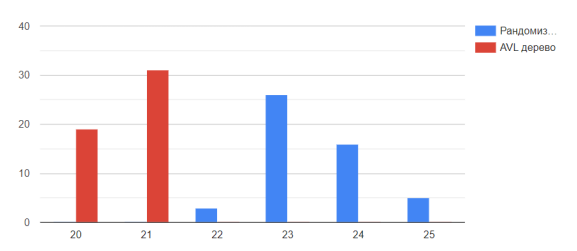
График для операции удаления:

# 

Также был построен график зависимости максимальной высоты от количества элементов в дереве.



Гистограмма среднего распределения высот веток в деревьях, где по х средняя высота веток в дереве, а по у – количество деревьев:



# Заключение

В ходе лабораторной работы были реализованы рандомизированное дерево поиска и AVL-дерево. Проанализировав результаты работы программы, можно сделать вывод, что у исследованных деревьев действительно одинаковая асимптотическая сложность, однако при выполнении операции вставки AVL-дерево оказалось эффективнее. Также можно отметить, что у AVL-дерева наименьшее максимальное значение длины дерева и его ветвей, чем у рандомизированного.