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 № 6

Выполнила студентка группы КС-36 Битарова Эмма Олеговна

Ссылка на репозиторий: https://github.com/MUCTR-IKT-CPP/

Приняли: Пысин Максим Дмитриевич

Дата сдачи: 27.11.2022

Оглавление

[Описание задачи. 2](#_Toc63548272)

[Описание метода/модели. 2](#_Toc63548273)

[Выполнение задачи. 2](#_Toc63548274)

[Заключение. 6](#_Toc63548275)

# Описание задачи.

В рамках лабораторной работы необходимо изучить и реализовать бинарное дерево поиска и его самобалансирующийся вариант в лице AVL дерева.

Для проверки анализа работы структуры данных требуется провести 10 серий тестов.

* В каждой серии тестов требуется выполнять 20 циклов генерации и операций. При этом первые 10 работают с массивом заполненным случайным образом, во второй половине случаев, массив заполняется в порядке возрастания значений индекса, т.е. является отсортированным по умолчанию.
* Требуется создать массив состоящий из 2^(10 + i) элементов, где i это номер серии.
* Массив должен быть помещен в оба вариант двоичных деревьев. При этом замеряется время затраченное на всю операцию вставки всего массива.
* После заполнения массива, требуется выполнить 1000 операций поиска по обоим вариантам дерева, случайного числа в диапазоне генерируемых значений, замерев время на все 1000 попыток и вычислив время 1 операции поиска.
* Провести 1000 операций поиска по массиву, замерить требуемое время на все 1000 операций и найти время на 1 операцию.
* После, требуется выполнить 1000 операций удаления значений из двоичных деревьев, и замерить время затраченное на все операции, после чего вычислить время на 1 операцию.
* После выполнения всех серий тестов, требуется построить графики зависимости времени затрачиваемого на операции вставки, поиска, удаления от количества элементов. При этом требуется разделить графики для отсортированного набора данных и заполненных со случайным распределением. Так же, для операции поиска, требуется так же нанести для сравнения график времени поиска для обычного массива.

# 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.

Процесс обхода дерева проще всего реализовать как рекурсивный процесс. Последовательно обследуя каждый узел дерева и посещая сначала левую его часть, затем правую. Стандартный порядок обхода выдаст отсортированный по возрастанию результат, его инверсия отсортированный по убыванию. Так же, можно использовать обход в ширину, тогда получится другой порядок обхода.

АВЛ дерево является обычным двоичным деревом поиска, следовательно его правое поддерево всегда меньше значения корня, а правое поддерево всегда больше. При это, при построении дерева мы руководствуемся правилом балансировки или перебалансировки: для любого узла дерева высота его правого поддерева отличается от высоты левого поддерева не более чем на единицу. Является доказанным, что при соблюдении этого правила высота дерева логарифмически зависит от количества элементов добавляемых в дерево, т.е. h = O(log(n)).

Узел AVL дерева, содержит:

* Ключ
* Высоту – от лепестков
* Левый потомок
* Правый потомок

По поводу высоты, можно учесть важный момент, классически атрибут высоты содержит не высоту, а разницу, которая может быть -1,0,1, так как в остальных случаях вызывается перебалансировка, но из за логарифмической природы высоты, можно смело хранить само значение, но тогда придется всегда рассчитывать фактор перебалансировки, который будет разницей между высотами правого и левого поддерева.

# Выполнение задачи.

Написано на языке С#. Были созданы 2 класса: AVL\_class и BinaryTree, содержащие в себе методы: Add, Find, Delete, Min\_Value (AVL), Max (Binary), Balance\_tree (Binary).

Результаты выполнения представлены в виде графиков.

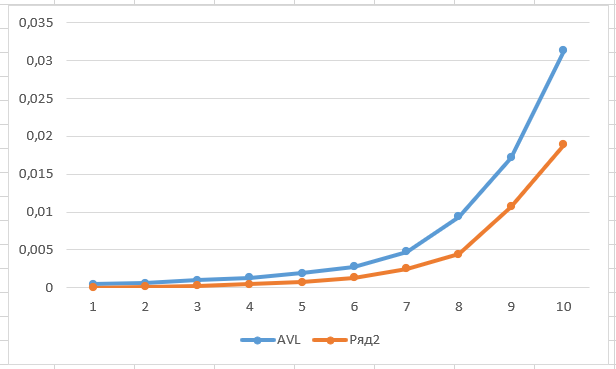


Рисунок 1.Добавление.

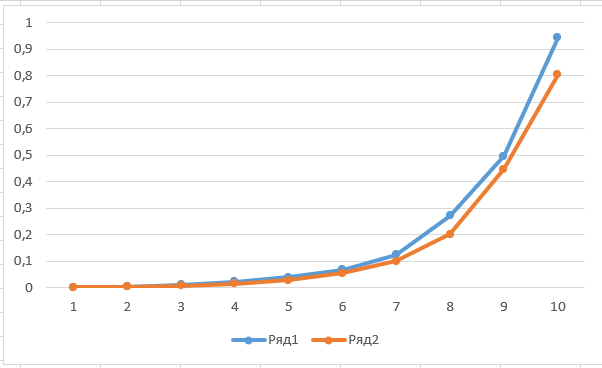


Рисунок 2. Поиск

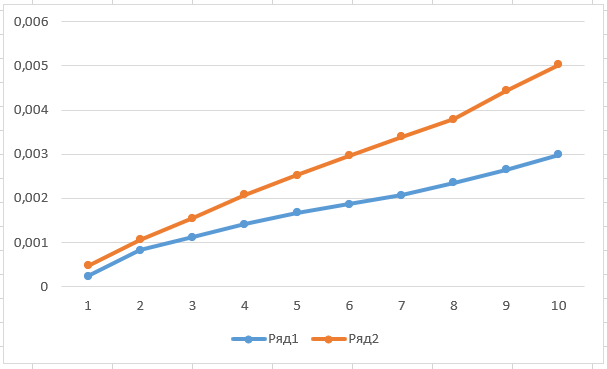


Рисунок 3. Удаление

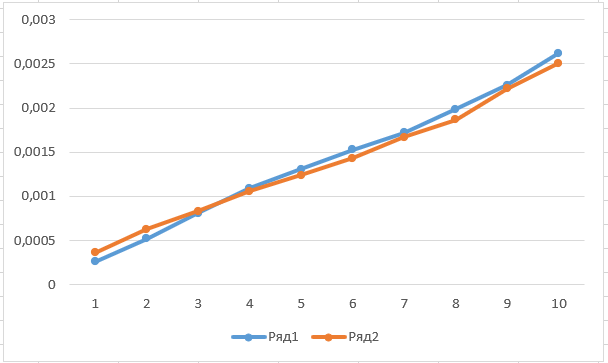


Рисунок 4. Добавление отсортированного.

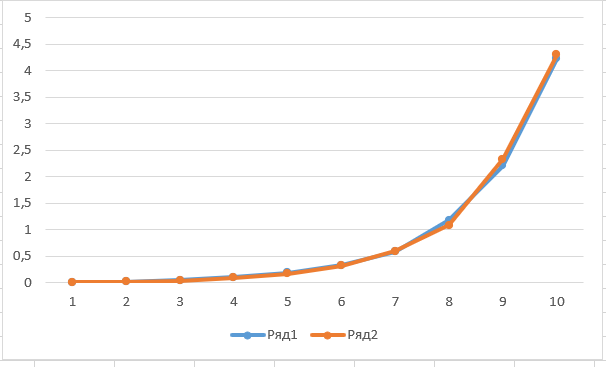


Рисунок 5. Поиск элемента отсортированного массива

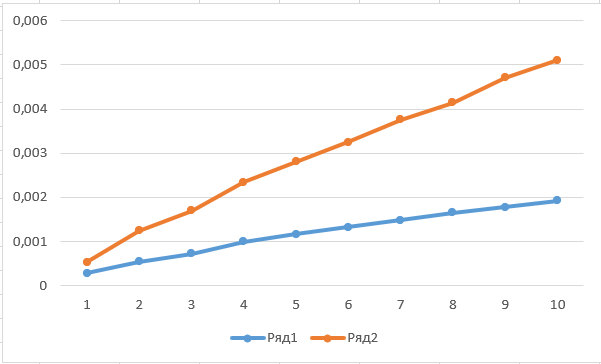
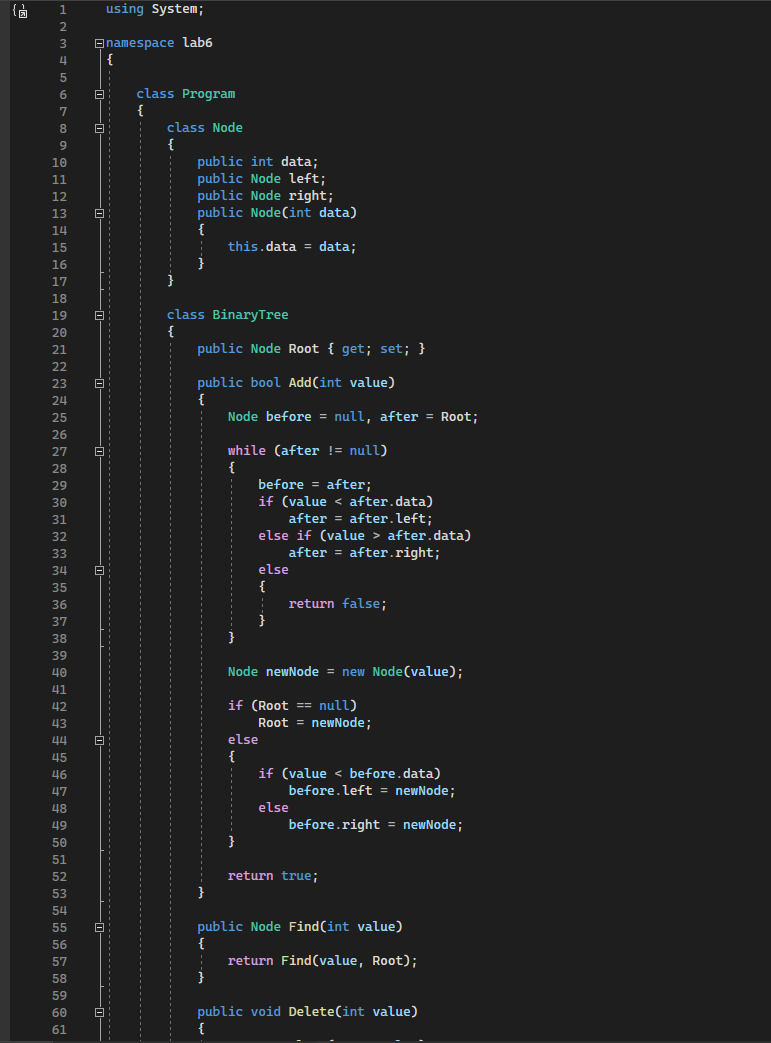
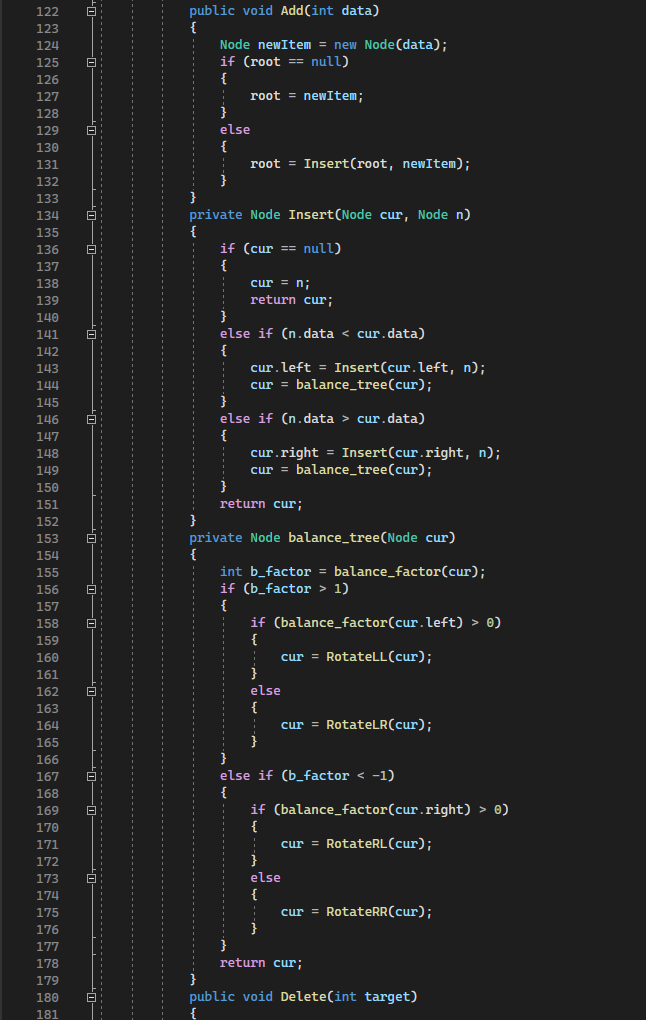


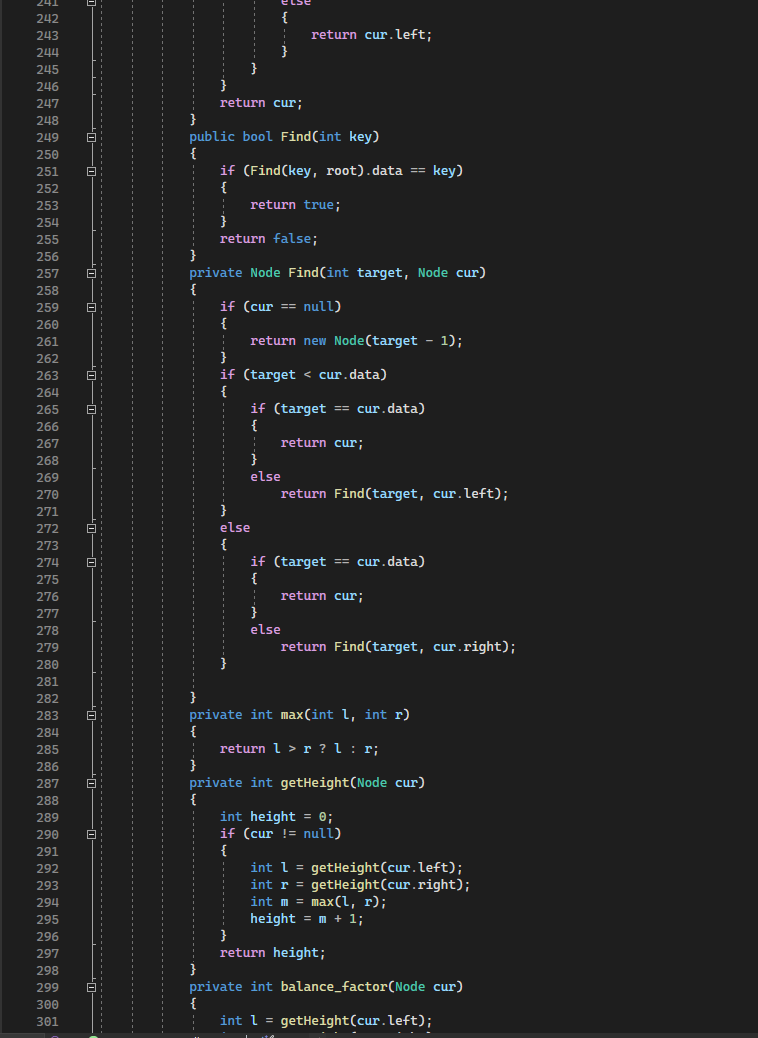
Рисунок 6. Удаление элемента отсортированного массива

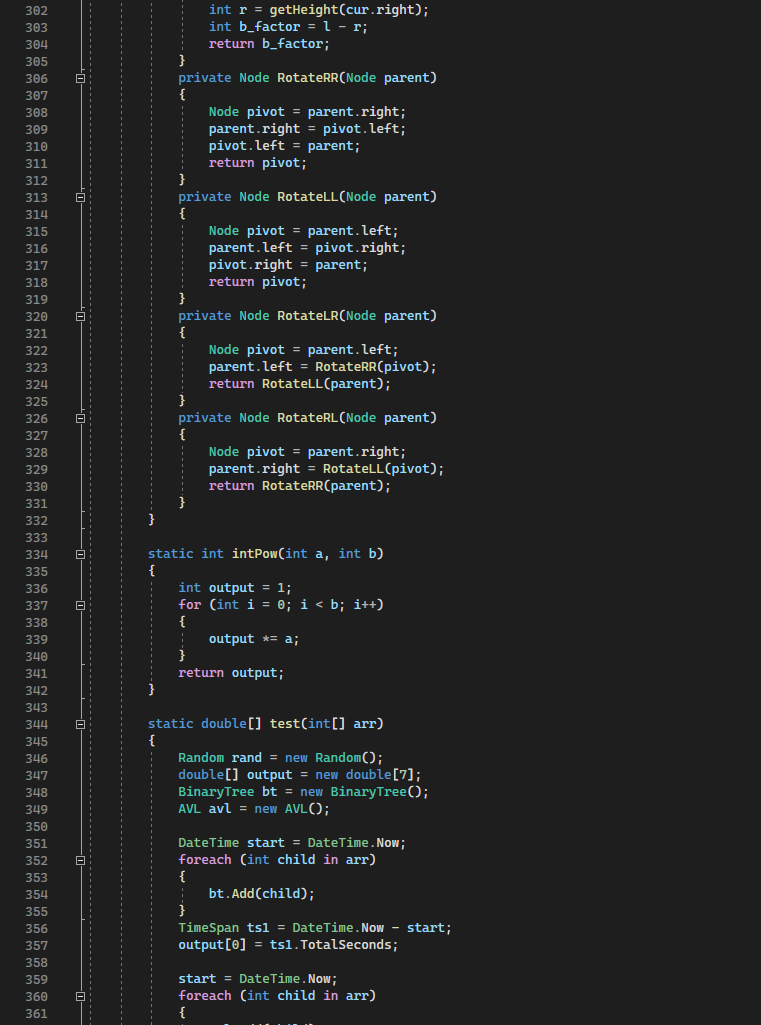


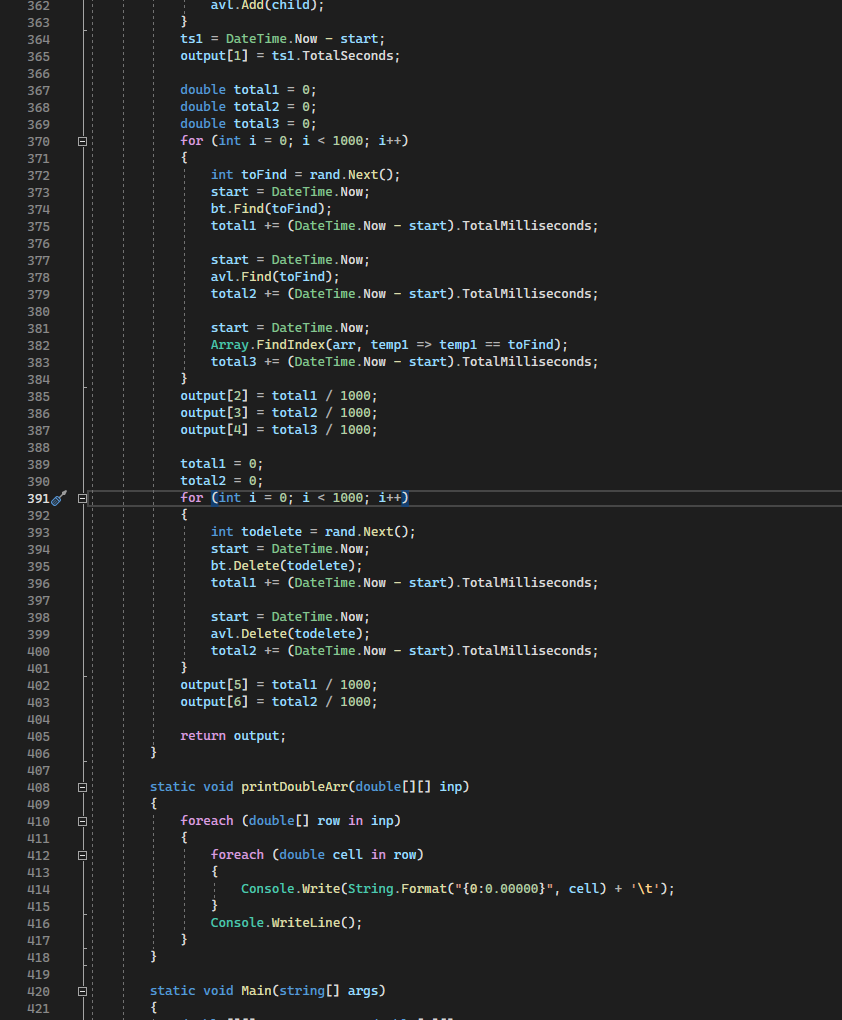


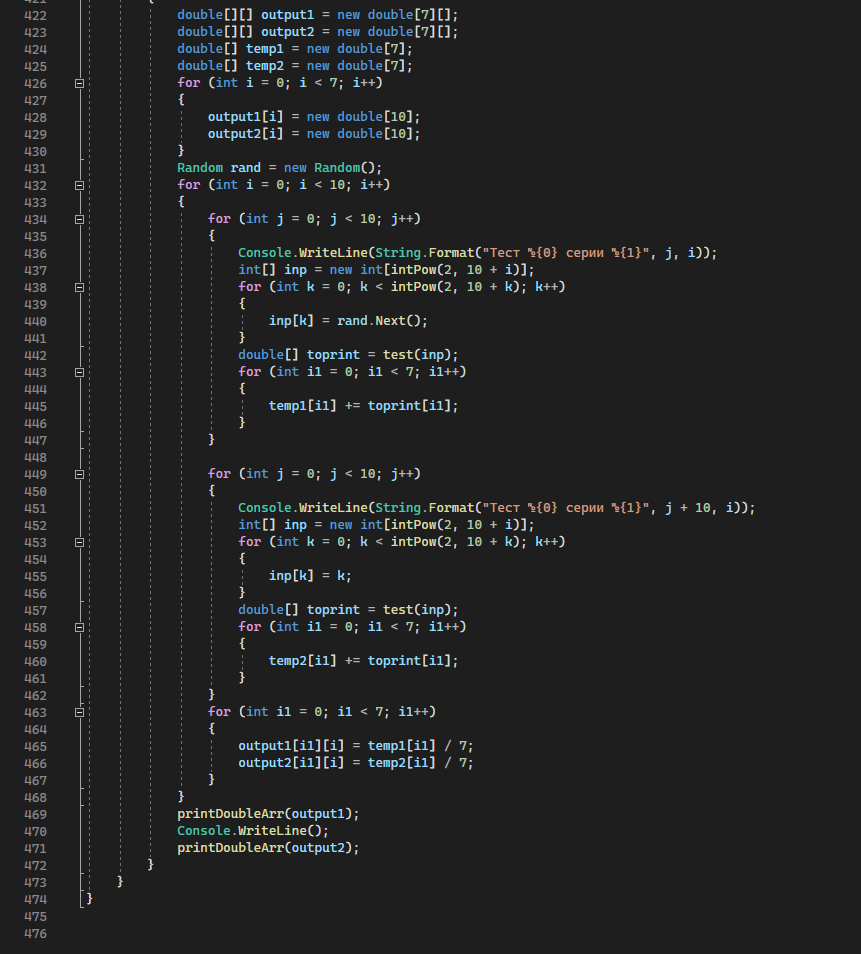












# Заключение.

Судя по графикам АВЛ дерево оказалось чаще медленнее чем бинарное дерево везде, кроме отсортированных данных. Если учитываться потери времени во время добавления/удаление элемента, то использование АВЛ деревьев не является удобным по скорости из-за частых балансировок.