Министерство науки и высшего образования Российской Федерации

#### ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

##### Кафедра «Математическое обеспечение и применение ЭВМ»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| «Утверждаю» | | | | |
| Зав. кафедрой «МО и ПЭВМ» | | | | |
|  | | | Макарычев П.П. | |
| « » |  |  | | 2021 |

### Пояснительная записка

к курсовой работе по дисциплине

«Структуры и алгоритмы обработки данных»

на тему: «Разработка библиотеки функций для поддержки работы со сбалансированными деревьями»

|  |  |
| --- | --- |
| Автор работы: | Кудашов А.С. |
| Направление бакалавриата | 09.03.04 («Программная инженерия») |
| Обозначение курсовой работы | ПГУ 09.03.04 - 03КР201.12 ПЗ |
| Группа | 20ВП1 |
| Руководитель работы | Самуйлов С.В, к.т.н., доцент |
| Работа защищена «\_\_» \_\_\_\_ 2021 г. | Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Пенза 2021 г.

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра «Математическое обеспечение и применение ЭВМ»

«УТВЕРЖДАЮ»

Зав. кафедрой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2021 г.

ЗАДАНИЕ

на курсовое проектирование по курсу

«Структуры и алгоритмы обработки данных»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студенту | Кудашову А.С. | | Группа | 20ВП1 |
| Тема проекта | | Разработка библиотеки функций для поддержки работы со сбалансированными деревьями | | |

Исходные данные (технические требования) на проектирование

|  |
| --- |
| 1. Должны быть разработаны следующие функции: |
| * 1. создать пустое сбалансированное дерево; |
| * 1. выполнить LL, RR, LR, RL-повороты; |
| * 1. добавить новое значение; |
| * 1. удалить значение; |
| * 1. модифицировать значение; |
| * 1. поиск значения; |
| * 1. обход слева направо; |
| * 1. визуализация сбалансированного дерева. |
| 1. Выполнить оценку временных характеристик алгоритма |
| оптимального бинарного поиска и алгоритма поиска в сбалансированном |
| дереве. Данные отобразить в табличной и графической формах. |
| 1. Выполнить оценку временных характеристик алгоритма |
| сортировки прямым включением и сортировки в сбалансированном |
| дереве. Данные отобразить в табличной и графической формах. |
| Для разработки приложений использовать по выбору одну из сред |
| визуального программирования Borland Delphi, Borland Builder C++, |
| Lazarus, MS Visual C++ и т.п. |
| Пользовательский интерфейс должен обеспечивать привычное и |
| удобное представление информации, простое и эффективное выполнение |
| основных функций приложения. |
| Программное обеспечение должно быть полностью отлажено и |
| протестировано, функционировать под управлением ОС Windows XP и |
| выше. |

Объем работы по курсу

1. Расчетная часть

|  |
| --- |
| 1. Анализ требований к разработке программного обеспечения |
| 1. Проектирование программы |
| 1. Разработка программного обеспечения |
| 1. Тестирование программного обеспечения |
|  |
|  |
|  |

2. Графическая часть

|  |
| --- |
| 1. Диаграмма вариантов использования |
| 1. Диаграмма классов |
| 1. Диаграмма деятельности |
| 1. Диаграмма компонентов |
|  |

3. Экспериментальная часть

|  |
| --- |
| 1. Подготовка набора тестовых данных |
| 1. Отладка и тестирование программного обеспечения |
|  |
|  |

Срок выполнения проекта по разделам

|  |  |  |
| --- | --- | --- |
| 1. Анализ требований к разработке ПО | к | 25.10.21 |
| 1. Проектирование программы | к | 10.11.21 |
| 1. Разработка программного обеспечения | к | 01.12.21 |
| 1. Тестирование программного обеспечения | к | 15.12.21 |
| 1. Оформление пояснительной записки | к | 28.12.21 |
| 1. Защита курсовой работы | к | 29.12.21 |
|  | к |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Дата выдачи задания | « 08» |  | сентября 2021 |
| Дата защиты проекта | « » |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Руководитель | |  | |
| Задание получил | | | «08» сентября 2021 г. |
| Студент |  | | |

# Реферат

Пояснительная записка содержит 64 листа, 21 рисунок, 5 таблиц,   
 7 использованных источников, 1 приложение.

СОРТИРОВКА, ПОИСК, АВЛ-ДЕРЕВЬЯ, БАЛАНСИРОВКА АВЛ-ДЕРЕВЬВ, СОРТИРОВКА МАСССИВА, ОПТИМАЛЬНЫЙ БИАРНЫЙ ПОИСК, СОРТИРОВКА ПРЯМЫМ ВКЛЮЧЕНИЕМ

Объектом исследования являются алгоритмы сортировки и поиска АВЛ-деревьев, а также алгоритмы оптимального бинарного поиска и сортировки прямым включением.

Целью курсовой работы является разработка приложения для сравнения и анализа алгоритмов внутренней сортировки.

Разработка проводилась на языке программирования С++ в среде программирования Visual Studio.

Осуществлено функциональное тестирование разработанного программного обеспечения, которое показало корректность его работы.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ПГУ 09.03.04 - 03КР201.12 ПЗ | | | | | |
|  |  |  |  |  |
| Изм. | Листт | № докум. | Подп. | Дата |
| Разраб. | | Кудашов А.С. |  |  | «Разработка библиотеки функций для поддержки работы со сбалансированными деревьями»  Пояснительная записка. | Лит. | | | Лист | Листов |
| Пров. | | Самуйлов С.В. |  |  |  |  |  | 4 | 64 |
|  | |  |  |  | Группа 20ВП1 | | | | |
|  | |  |  |  |
| Н. контр. | |  |  |  |
| Утв. | |  |  |  |

Оглавление

[Введение 6](#_Toc87130174)

[1. Постановка задачи и анализ предметной области 7](#_Toc87130175)

[1.1. Основные понятия и определения 7](#_Toc87130176)

[1.2. Постановка задачи на разработку программы 9](#_Toc87130177)

[1.3. Анализ требований 9](#_Toc87130178)

[1.4. Технология разработки программного обеспечения 14](#_Toc87130179)

[2. Проектирование программы 16](#_Toc87130180)

[2.1. Модель интерфейса 16](#_Toc87130183)

[2.2. Проектирование структур данных 17](#_Toc87130184)

[2.3. Структура программного обеспечения 18](#_Toc87130185)

[3. Реализация программы 24](#_Toc87130186)

[3.1. Кодирование 24](#_Toc87130187)

[3.2. Диаграмма компонентов 24](#_Toc87130188)

[4. Тестирование программы 26](#_Toc87130189)

[4.1. Виды тестирования программных средств 26](#_Toc87130190)

[4.2. Функциональное тестирование программы 29](#_Toc87130191)

[5. Анализ результатов 34](#_Toc87130192)

[Заключение 35](#_Toc87130193)

[Список использованных источников 36](#_Toc87130194)

[Приложение А. Листинг программы «coursework.exe» 37](#_Toc87130195)

# Введение

В современном мире при разработке программных средств зачастую используется такая структура данных, как бинарное дерево. Одним из недостатков данного типа является то, что высота различных поддеревьев может значительно отличаться, что влечет за собой уменьшение скорости поиска значений в дереве. Заметным улучшением является допущение, что для каждой его вершины высота двух ее поддеревьев различается не более чем на 1. Такой вид дерева называется сбалансированным (или АВЛ-деревом). [2]

Деревья широко применяются в разработке программных средств. Самым ярким примером может служить файловая система. Поэтому выбор вида дерева, который будет использован, может значительно сказаться на скорости работы приложения.

Целями данной практической работы являются реализация функций работы со сбалансированными деревьями, выявление того, какие методы сортировки и поиска быстрее: методы работы с линейными структурами или со сбалансированными деревьями.

Для достижения поставленной цели необходимо:

1. реализовать методы работы с АВЛ-деревьями;
2. измерить время работы алгоритмов сортировки прямым включением и сортировки в АВЛ-дереве;
3. измерить время работы алгоритмов оптимального бинарного поиска и поиска в АВЛ-дереве;
4. построить по полученным данным таблицы и диаграммы;
5. сравнить результаты;
6. проанализировать результаты.

# Постановка задачи и анализ предметной области

### Основные понятия и определения

АВЛ-дерево — сбалансированное по высоте [двоичное дерево поиска](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%D0%BF%D0%BE%D0%B8%D1%81%D0%BA%D0%B0): для каждой его вершины высота её двух поддеревьев различается не более чем на 1.

АВЛ — аббревиатура, образованная первыми буквами фамилий создателей (советских учёных) [Адельсон-Вельского Георгия Максимовича](https://ru.wikipedia.org/wiki/%D0%90%D0%B4%D0%B5%D0%BB%D1%8C%D1%81%D0%BE%D0%BD-%D0%92%D0%B5%D0%BB%D1%8C%D1%81%D0%BA%D0%B8%D0%B9,_%D0%93%D0%B5%D0%BE%D1%80%D0%B3%D0%B8%D0%B9_%D0%9C%D0%B0%D0%BA%D1%81%D0%B8%D0%BC%D0%BE%D0%B2%D0%B8%D1%87) и [Ландиса Евгения Михайловича](https://ru.wikipedia.org/wiki/%D0%9B%D0%B0%D0%BD%D0%B4%D0%B8%D1%81,_%D0%95%D0%B2%D0%B3%D0%B5%D0%BD%D0%B8%D0%B9_%D0%9C%D0%B8%D1%85%D0%B0%D0%B9%D0%BB%D0%BE%D0%B2%D0%B8%D1%87). Также они выявили зависимость между высотой дерева и количество узлов: . [1]

Балансировкой вершины АВЛ-дерева называется операция, которая в случае разницы высот левого и правого поддеревьев, равной 2, изменяет связи предок-потомок в поддереве данной вершины так, что разница не превышает 1, иначе ничего не меняет. Указанный результат получается вращениями поддерева данной вершины. [2]

Используются 4 типа вращений:

* Малое левое вращение (LL). Используется, когда (высота b-поддерева - высота L) = 2 и высота С <= высота R. (Рис. 1)

[](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:AVL_LR.GIF)

Рисунок 1 - Малое левое вращение.

* Большое левое вращение (LR). Используется, когда (высота b-поддерева - высота L) = 2 и высота c-поддерева>высота R. (Рис. 2)



Рисунок 2 - Большое левое вращение.

* Малое правое вращение (RR). Используется, когда (высота b-поддерева - высота R) = 2 и высота С <= высота L. (Рис. 3)



Рисунок 3 - Малое правое вращение.

* Большое правое вращение (RL). Используется, когда (высота b-поддерева - высота R) = 2 и высота c-поддерева>высота L. (Рис. 4)



Рисунок 4 - Большое правое вращение.

В каждом случае достаточно просто доказать то, что операция приводит к нужному результату и что полная высота уменьшается не более чем на 1 и не может увеличиться. Также можно заметить, что большое левое вращение — это композиция правого малого вращения и левого малого вращения, большое правое - левого малого и правого малого вращений.

Сортировка методом прямого включения, обычно применяется для массивов, не содержащих повторяющихся элементов. Она производится по шагам. На k - м шаге считается, что часть массива, содержащая первые k-1 элементов, уже упорядочена, то есть а [1] ≤ а [2] ≤ ... ≤ a [k-1].

Далее необходимо взять k - й элемент и подобрать для него такое место в отсортированной части массива, чтобы после его вставки упорядоченность не нарушалась, то есть надо найти такое j (1 ≤ j ≤ k -1), что а[j] ≤ a[k] < a[j+1]. Затем вставить элемент а [k] на найденное место.

С каждым шагом отсортированная часть массива увеличивается. Для выполнения полной сортировки потребуется выполнить n-1 шаг. [3]

### Постановка задачи на разработку программы

Необходимо разработать библиотеку функций для поддержки работы со сбалансированными деревьями. Постановка задачи состоит из трех пунктов:

1. Разработка следующих функций:
   1. создать пустое сбалансированное дерево;
   2. выполнить LL-поворот;
   3. выполнить RR-поворот;
   4. выполнить LR-поворот;
   5. выполнить RL-поворот;
   6. добавить новое значение;
   7. удалить значение;
   8. модифицировать значение;
   9. поиск значения;
   10. обход слева направо;
   11. визуализация сбалансированного дерева.
2. Выполнить оценку временных характеристик работы алгоритма оптимального бинарного поиска и алгоритма поиска в сбалансированном дереве. Данные отобразить в табличной и графической формах.
3. Выполнить оценку временных характеристик работы алгоритма сортировки прямым включением и сортировки с помощью сбалансированного дерева. Данные отобразить в табличной и графической формах.

### Анализ требований

#### Требования к интерфейсу пользователя

Пользовательский интерфейс – это комплекс средств для взаимодействия пользователя с технической системой.

Приложение «Сбалансированные деревья» должно предоставлять пользователю удобный интерфейс, позволяющий:

* + 1. задавать размеры массива исходных данных;
    2. взаимодействовать со сбалансированным АВЛ-деревом:
       1. добавлять новые элементы;
       2. удалять заданные элементы;
       3. изменять значения элементов;
       4. визуализировать АВЛ-дерево;
    3. сравнивать время работы алгоритмов сортировки и поиска;
    4. отображать результаты работы алгоритмов в виде таблицы;
    5. отображать результаты работы алгоритмов в графической форме.

#### Требования к структуре данных

В разрабатываемой программе используются следующие структуры данных:

1. массив (Array);
2. сбалансированное дерево.

Рассмотрим требования к каждому из них по отдельности.

Массив - структура данных в виде набора компонентов (элементов массива), расположенных в памяти непосредственно друг за другом. При этом доступ к отдельным элементам массива осуществляется с помощью индексации, то есть через ссылку на массив с указанием номера (индекса) нужного элемента. [4] Логическая структура одномерного массива приведена на рисунке 5.



Рисунок 5 – Логическая структура массива.

Основные операции:

1. установление размерности массива;
2. запись элемента в массив по индексу;
3. упорядочивание элементов массива;
4. очищение массива и т.д.

Сбалансированное дерево представляет собой дерево, для каждой его вершины высота её двух поддеревьев различается не более чем на 1

Основные операции:

1. выполнение LL, LR, RR, RL-повороты;
2. добавление нового значения;
3. удаление значения;
4. изменение значения;
5. поиск значения;
6. обход слева направо;
7. визуализация.

#### Требования к программным средствам

Разрабатываемая программа должна сравнивать время работы алгоритмов оптимального бинарного поиска, сортировки прямым включением и алгоритмов работы со сбалансированными деревьями. Анализ задания на разработку позволяет выделить следующие варианты использования (рис. 6).

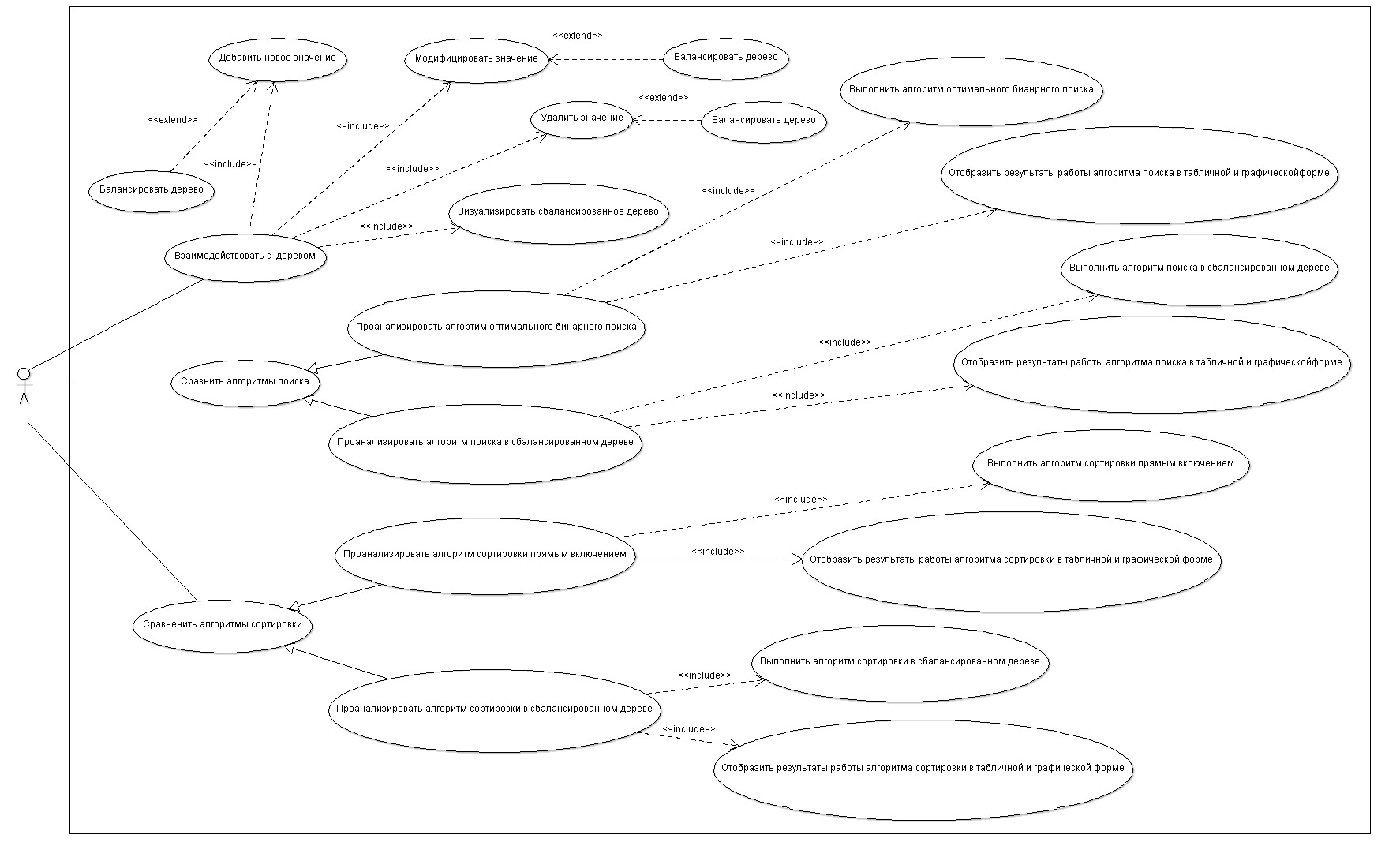


Рисунок 6 - Диаграмма вариантов использования.

Опишем спецификацию нескольких прецедентов. Рассмотрим спецификацию прецедента «Выполнить алгоритм сортировки в сбалансированном дереве» (Таблица 1):

Таблица 1- Спецификация прецедента «Выполнить алгоритм сортировки в сбалансированном дереве»

|  |
| --- |
| Прецедент: Выполнить алгоритм сортировки в сбалансированном дереве |
| ID: 1 |
| Краткое описание: Выполнить алгоритм сортировки массива с помощью преобразования в сбалансированное АВЛ-дерево |
| Действующие лица: пользователь |
| Предусловия: Пользователь нажал «Отсортировать с помощью сбалансированного АВЛ-дерева» |
| Основной поток:   1. Пользователь выбирает размерность массива 2. Система заполняет массив случайными числами 3. Система выстраивает двоичное дерево 4. Система собирает результирующий массив путём обхода узлов в необходимом порядке следования ключей 5. Система выдает результаты сортировки в табличной и графической форме |
| Постусловия: Сформирована таблица и нарисован график |

Рассмотрим спецификацию прецедента «Взаимодействовать с деревом» (Таблица 2):

Таблица 2- Спецификация прецедента «Взаимодействовать с деревом»

|  |
| --- |
| Прецедент: Взаимодействовать с деревом |
| ID: 2 |
| Краткое описание: Строится АВЛ-дерево на основе действий пользователя |
| Действующие лица: пользователь |
| Предусловия: Пользователь нажал «Взаимодействовать» |
| Основной поток:   1. Пользователь вводит первое значение 2. Если пользователь вводит значения    1. Система добавляет значение в дерево    2. Если разбалансировка       1. Дерево балансируется 3. Если пользователь удаляет значения    1. Система удаляет значение из дерева    2. Если разбалансировка       1. Дерево балансируется 4. Если пользователь изменяет значения    1. Система изменяет значение в дереве    2. Если разбалансировка       1. Дерево балансируется 5. Система строит дерево на экране |
| Постусловия: Дерево построено |

Рассмотрим спецификацию прецедента «Выполнить алгоритм сортировки прямым включением» (Таблица 3):

Таблица 3- Спецификация прецедента «Выполнить алгоритм сортировки прямым включением»

|  |
| --- |
| Прецедент: Выполнить алгоритм сортировки прямым включением |
| ID: 3 |
| Краткое описание: Выполнить алгоритм сортировки массива методом прямого включения |
| Действующие лица: пользователь |
| Предусловия: Пользователь нажал «Отсортировать методом сортировки прямым включением» |
| Основной поток:   1. Пользователь выбирает размерность массива 2. Система заполняет массив случайными числами 3. Система сортирует массив методом прямого включения 4. Система выдает результаты сортировки в табличной и графической форме |
| Постусловия: Сформирована таблица и нарисован график |

### Технология разработки программного обеспечения

Для разработки ПО достаточно часто используются следующие технологии разработки:

1. «Waterfall model» - каскадная методология
2. RAD – для разработки интерфейса

Каскадная модель (англ. waterfall model) — модель процесса разработки программного обеспечения, в которой процесс разработки выглядит как поток, последовательно проходящий фазы анализа требований, проектирования, реализации, тестирования, интеграции и поддержки.

В каскадной модели этапы проектирования постепенно проходят в следующем порядке:

1. проектирование;
2. дизайн;
3. кодирование;
4. тестирование;
5. поддержка.

RAD — концепция создания средств разработки программных продуктов, уделяющая особое внимание быстроте и удобству программирования, созданию технологического процесса, позволяющего программисту максимально быстро создавать компьютерные программы. [5] Практическое определение: RAD — это жизненный цикл процесса проектирования, созданный для достижения более высокой скорости разработки и качества ПО, чем это возможно при традиционном подходе к проектированию. С конца XX века RAD получила широкое распространение и одобрение. Концепцию RAD также часто связывают с концепцией визуального программирования.

Типичная модель RAD может быть разделена на пять этапов:

1. Планирование и анализ требований.
2. Проектирование архитектуры проекта.
3. Разработка и программирование.
4. Тестирование.
5. Развертывание и обслуживание.

* **Планирование и анализ требований -** этот этап является одним из наиболее важных этапов. Здесь исходные требования собраны и проанализированы должным образом. Помните, что правильное понимание требований очень необходимо, чтобы готовый продукт соответствовал ожиданиям.
* **Проектирование архитектуры проекта.** После того, как требования собраны, следующим фокусом будет разработка архитектуры проекта. Архитектура проекта должна быть достаточно гибкой, чтобы можно было легко добавлять новые файлы и папки.
* **Разработка и программирование -** после того, как архитектура спроектирована, следующей важной задачей является разработка проекта. Этот шаг включает в себя написание куч кода, чтобы получить недопустимое состояние продукта.
* **Тестирование** - этап тестирования включает в себя тестирование разработанного продукта. Существует команда, которая занимается надлежащим тестированием разработанного продукта.
* **Развертывание и обслуживание.** После завершения тестирования продукт может быть развернут на сервере. Развернутый проект обычно требует обслуживания и, возможно, добавления нескольких дополнительных функций. [6]

Технология быстрой разработки приложений (RAD) позволяет обеспечить:

* интерфейс, устраивающий пользователя
* лёгкую адаптируемость проекта к изменяющимся требованиям
* простоту развития функциональности системы.

При выполнении курсовой работы было принято решение использовать технологию разработки RAD.

# Проектирование программы



### Модель интерфейса

Работа приложения начинается с запуска страницы алгоритмов сортировки и поиска. Модель её интерфейса представлена на рисунке 7.

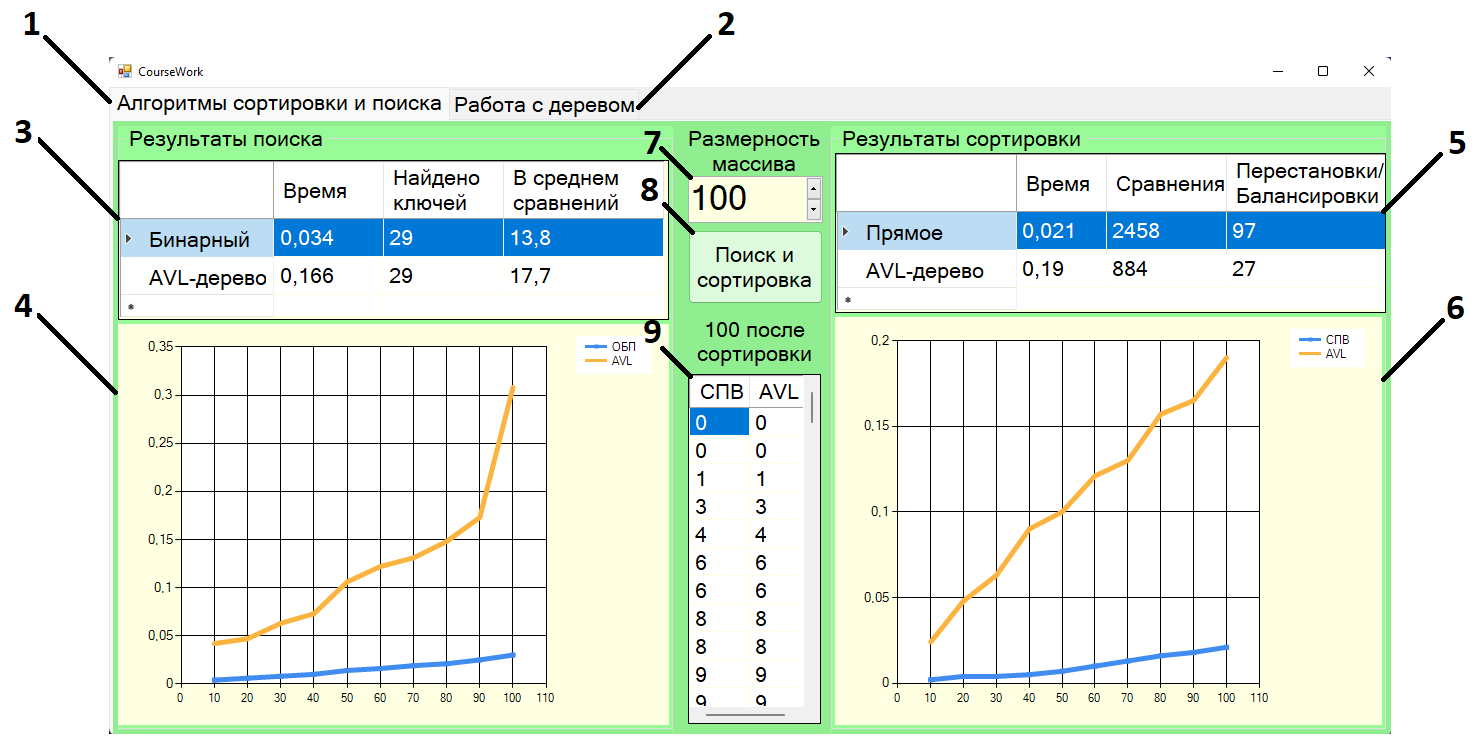


Рисунок 7 - Модель интерфейса страницы алгоритмов сортировки и поиска.

Вторая страница приложения предназначена для демонстрации возможностей взаимодействия с АВЛ-деревом и его визуализации. Модель её интерфейса представлена на рисунке 8.

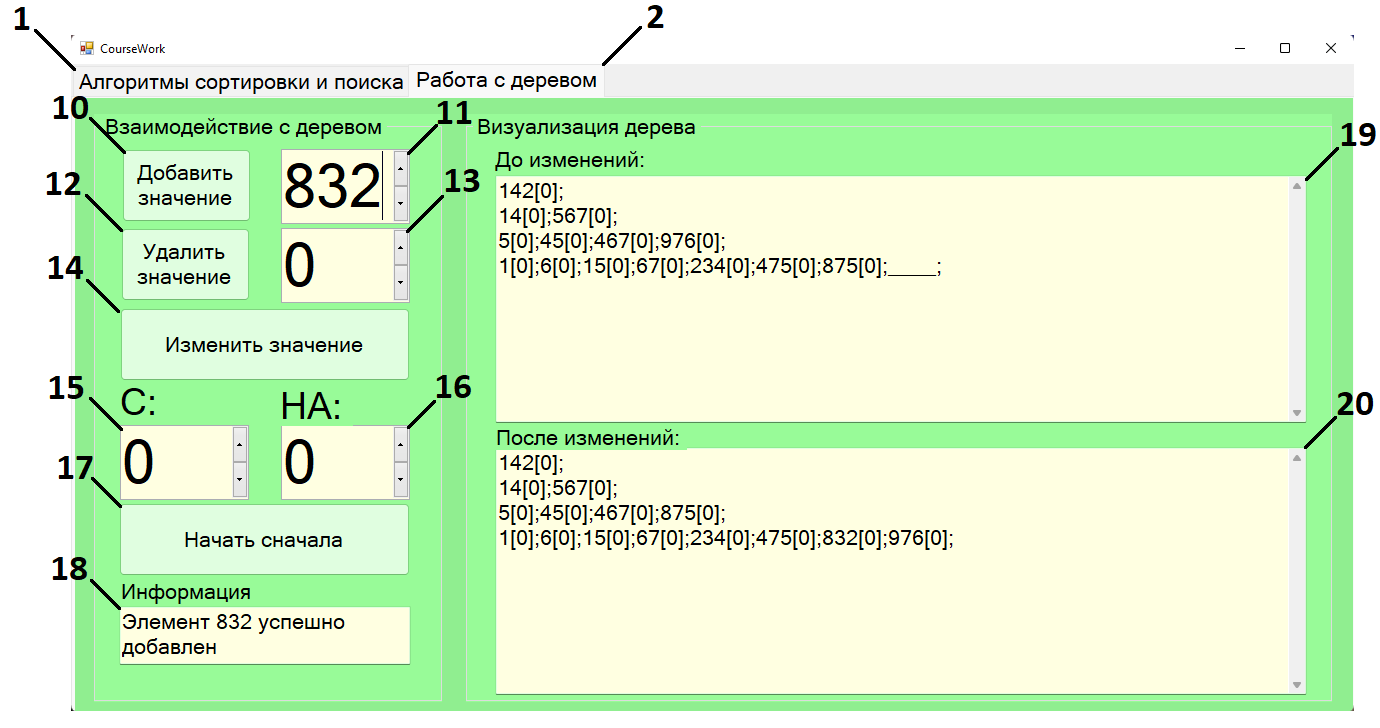


Рисунок 8 - Модель интерфейса страницы для работы с АВЛ-деревом.

Обозначения на рисунках 7 и 8:

1. страница для демонстрации сортировки и поиска;
2. страница для демонстрации алгоритмов работы АВЛ-деревьев;
3. таблица с характеристиками алгоритмов поиска;
4. график демонстрации зависимости времени работы алгоритмов поиска от количества элементов;
5. таблица с характеристиками алгоритмов сортировки;
6. график демонстрации зависимости времени работы алгоритмов сортировки от количества элементов;
7. поле ввода размерности массива;
8. кнопка выполнения сортировки и поиска;
9. таблица демонстрации первых 100 элементов после выполнения сортировки;
10. кнопка добавления значения в АВЛ-дерево;
11. поле ввода значения для добавления в АВЛ-дерево;
12. кнопка удаления значения из АВЛ-дерева;
13. поле ввода значения для удаления из АВЛ -дерева;
14. кнопка изменения значения в АВЛ -дереве;
15. поле ввода исходного значения при изменении в АВЛ -дереве;
16. поле ввода конечного значения при изменении в АВЛ -дереве;
17. кнопка создания нового дерева;
18. поле вывода результатов работы алгоритма;
19. поле вывода АВЛ -дерева до внесения именений;
20. поле вывода АВЛ -дерева после внесения изменений.

### Проектирование структур данных

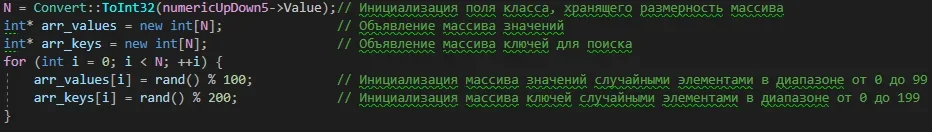
Основной структурой данных в курсовой работе является неупорядоченный одномерный массив. Код объявления и инициализации массивов приведён на рисунке 9. Освобождение памяти массивов представлено на рисунке 10.

Рисунок 9 – Объявление и инициализация одномерных неупорядоченных массивов.



Рисунок 10 – Освобождение памяти одномерных неупорядоченных массивов.

### Структура программного обеспечения

Структура разрабатываемого приложения представлена на диаграмме классов (рисунок 11).

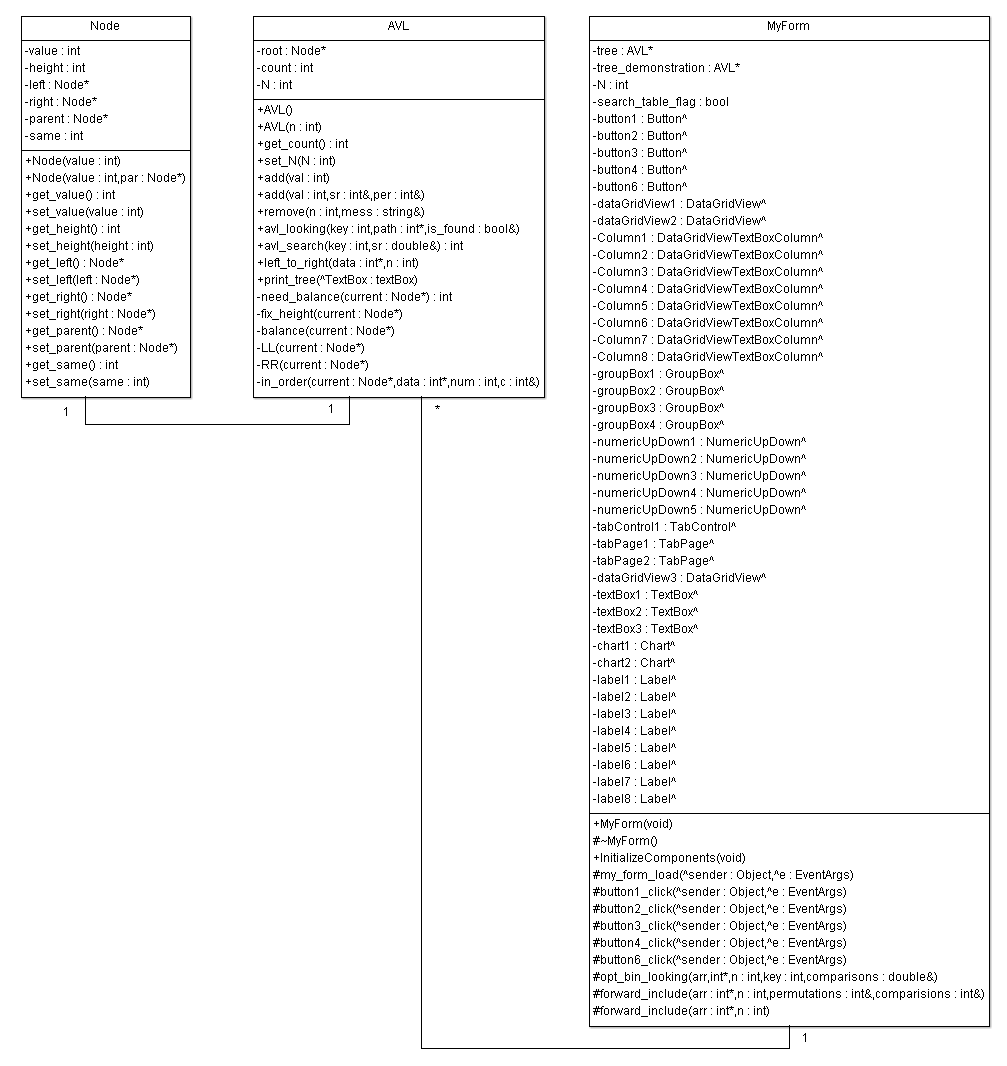


Рисунок 11 – Диаграмма классов

На приведенной диаграмме классы MyForm и AVL, а также классы Node и AVL находятся в отношении ассоциации.

Рассмотрим спецификации всех классов:

* спецификация класса MyForm – таблица 1;
* спецификация класса AVL – таблица 2;
* спецификация класса Node – таблица 3.

Таблица 1 – Спецификация класса MyForm

|  |  |
| --- | --- |
| Название класса: | MyForm |
| Назначение класса: | Класс для создания графического интерфейса и настройки взаимодействия с ним |
| Поля класса: | -tree: AVL\* – дерево для поиска и сортировки;  -tree\_demonstration: AVL\* – дерево для демонстрации работы АВЛ -деревьев;  -N: int – размерность массивов;  -search\_table\_flag: bool – информация о сгенерированности таблицы;  -button1: Button^ – кнопка, при нажатии на которую происходит добавление элемента в дерево;  -button2: Button^ – кнопка, при нажатии на которую происходит удаление элемента из дерева;  -button3: Button^ – кнопка, при нажатии на которую происходит изменение элемента в дереве;  -button4: Button^ – кнопка, при нажатии на которую происходит поиск и сортировка;  -button6: Button^ – кнопка, при нажатии на которую происходит создание нового дерева для демонстрации;  -dataGirdView1: DataGirdView^ – таблица с характеристиками поиска;  -dataGirdView2: DataGirdView^ – таблица с характеристиками сортировки;  -dataGirdView3: DataGirdView^ – таблица с первыми 100 элементами после сортировки;  -Column1, Column2, Column3: DataGirdViewTextBoxColumn^ – столбцы таблицы dataGirdView1;  -Column4, Column5, Column6: DataGirdViewTextBoxColumn^ – столбцы таблицы dataGirdView2;  -Column7, Column8: DataGirdViewTextBoxColumn^ – столбцы таблицы dataGirdView3;  - groupBox1, groupBox2, groupBox3, groupBox4: GroupBox^ – элементы объединения групп интерфейса;  -numericUpDown1: NumericIpDown^ – поле ввода добавляемого в дерево значения;  -numericUpDown2: NumericIpDown^ – поле ввода удаляемого из дерева значения;  -numericUpDown3: NumericIpDown^ – поле ввода исходного значения при изменении в АВЛ-дереве;  -numericUpDown4: NumericIpDown^ – поле ввода конечного значения при изменении в АВЛ-дереве;  -numericUpDown5: NumericIpDown^ – поле ввода размерности массива;  -tabControl1: TabControl^ – контейнер с вкладками;  -tabPage1: TabPage^ – вкладка сортировки и поиска;  -tabPage2: TabPage^ – вкладка демонстрации алгоритмов работы АВЛ-деревьев;  -textBox1: TextBox^ – поле вывода АВЛ-дерева до внесения именений;  -textBox2: TextBox^ – поле вывода АВЛ-дерева после внесения именений;  -textBox3: TextBox^ – поле вывода результатов работы алгоритма;  -chart1: Chart^ – график демонстрации зависимости времени работы алгоритмов поиска от количества элементов;  -chart2: Chart^ – график демонстрации зависимости времени работы алгоритмов сортировки от количества элементов;  -label1, label2, label3, label4, label5, label6, label7, label8: Label^ – надписи на форме. |
| Методы класса: | +MyForm(void) – конструктор класса;  #~MyForm () – деструктор класса;  +InitializeComponents(void) – создание графического интерфейса;  #button1\_click(Object, EventArgs) – обработка клика кнопки добавления значения в АВЛ-дерево;  #button2\_click(Object, EventArgs) – обработка клика кнопки удаления значения из АВЛ-дерева;  #button3\_click(Object, EventArgs) – обработка клика кнопки изменения значения в АВЛ-дереве;  #button4\_click(Object, EventArgs) – обработка клика кнопки выполнения сортировки и поиска;  #button6\_click(Object, EventArgs) – обработка клика кнопки создания нового дерева;  #opt\_bin\_looking(int\*, int, int, double&) – оптимальный бинарный поиск;  #forward\_include(int\*, int, int&, int&) – сортировка прямым включением (с получением характеристик);  #forward\_include(int\*, int) – сортировка прямым включением (без получения характеристик). |

Таблица 2 – Спецификация класса AVL

|  |  |
| --- | --- |
| Название класса: | AVL |
| Назначение класса: | Класс сущности АВЛ-дерева |
| Поля класса: | -root: Node\* – корень дерев;  -count: int – количество элементов в дереве на текущий момент;  -N: int – размерность дерева. |
| Методы класса: | +AVL() – конструктор класса;  +AVL(int) – конструктор класса;  +get\_count() – получение значения поля count;  +set\_N(int) – изменение значение поля N;  +add(int) – добавление элемента в дерево (без получения характеристик);  +add(int, int&, int&) – добавление элемента в дерево (с получением характеристик);  +remove(int, string&) – удаление элемента из дерева;  +avl\_looking(int, int\*, bool&) – поиск ключа (с сохранением пути до него);  +avl\_search(int, double&) – поиск ключа (без сохранения пути до него);  +left\_to\_right(int\*, int) – обход слева направо;  +print\_tree(TextBox) – вывод дерева;  -need\_balance(Node\*) – проверка необходимости балансировки;  -fix\_height(Node\*) – восстановление значения высоты;  -balance(Node\*) – балансировка;  -LL(Node\*) – левый поворот;  -RR(Node\*) – правый поворот;  -in\_order(Node\*, int\*, int, int&) – рекурсивный обход. |

Таблица 3 – Спецификация класса Node

|  |  |
| --- | --- |
| Название класса: | Node |
| Назначение класса: | Класс узла АВЛ-дерева |
| Поля класса: | -value: int – значение в узле  -height: int – высота узла  -left: Node\* – левое поддерево  -right: Node\* – правое поддерево  -parent: Node\* – родительский элемент узла  -same: int – количество элементов с повторяющимся значением |
| Методы класса: | +Node(int) – конструктор класса;  +Node (int, Node\*) – конструктор класса;  +get\_value() – получение значения поля value;  +set\_value(int) – изменение значения поля value;  +get\_ height () – получение значения поля height;  +set\_ height (int) – изменение значения поля height;  +get\_ left () – получение значения поля left;  +set\_ left (Node\*int) – изменение значения поля left;  +get\_ right () – получение значения поля right;  +set\_right (Node\*) – изменение значения поля right;  +get\_ parent () – получение значения поля parent;  +set\_parent (Node\*) – изменение значения поля parent;  +get\_same () – получение значения поля same;  +set\_same (int) – изменение значения поля same; |

# Реализация программы

### Кодирование

Код программы приведен в приложении 1.

Рассмотрим диаграмму деятельности метода «avl\_search» (рисунок 12). Как видно из диаграммы, в методе проверяется соответствие значения в текущем узле ключу (больше, меньше или равно), и в зависимости от результата либо завершается цикл проверок, либо текущий узел передвигается к следующему. В процессе выполнения алгоритма происходит подсчет количества сравнений. Возвращается число 1 или 0 в зависимости от того, найдено соответствие ключу или нет.

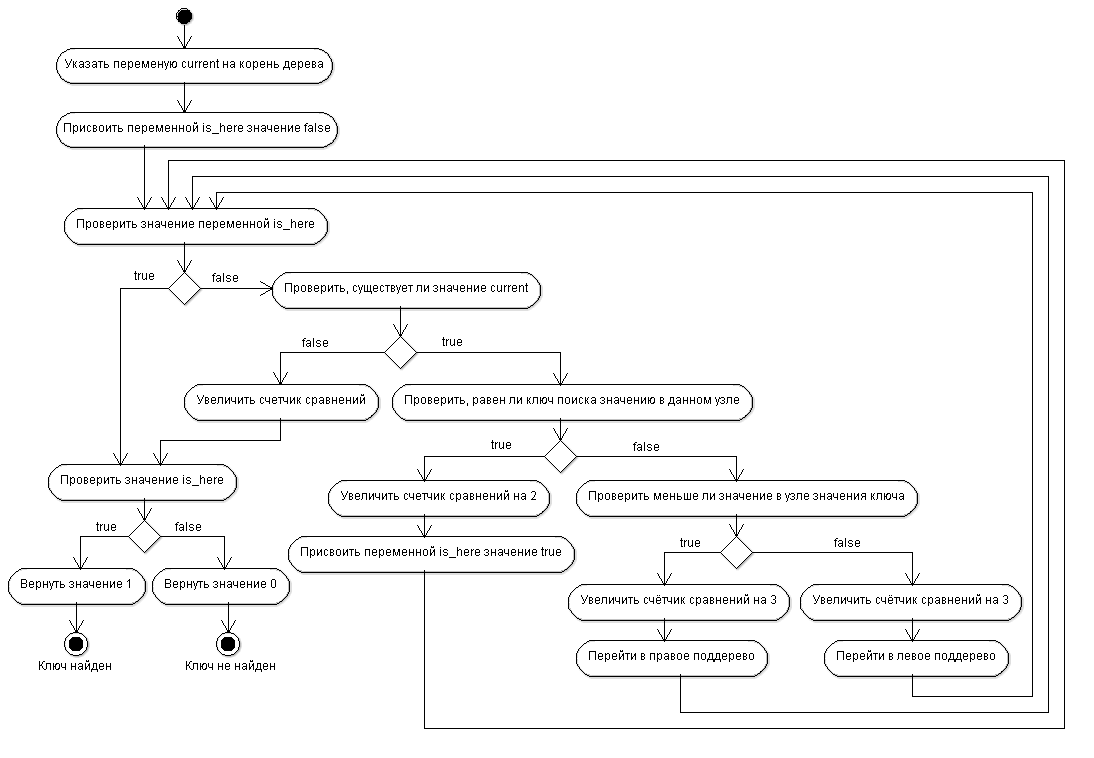


Рисунок 12 – Диаграмма деятельности метода «avl\_search».

### Диаграмма компонентов

Модули программы и взаимоотношения между ними представлены на диаграмме компонентов (рисунок 12).

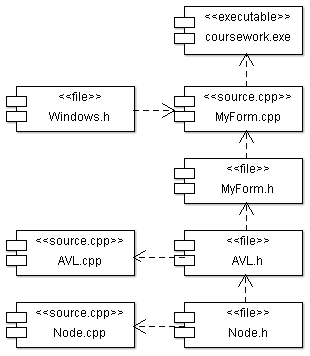


Рисунок 13 – Диаграмма компонентов.

Описание компонентов приведено в таблице 4.

Таблица 4 - Компоненты

|  |  |
| --- | --- |
| Компоненты | Назначение |
| coursework.exe | Исполняемый файл приложения |
| MyForm.cpp | Исходный файл класса MyForm |
| MyForm.h | Заголовочный файл класса MyForm |
| Windows.h | Системный заголовочный файл |
| AVL.h | Заголовочный файл класса AVL |
| AVL.cpp | Исходный файл класса AVL |
| Node.h | Заголовочный файл класса Node |
| Node.cpp | Исходный файл класса Node |

1. Тестирование программы

### Виды тестирования программных средств

Тестирование программного обеспечения – это процесс испытания программного продукта с целью проверки соответствия между реальным и ожидаемым поведением программы, также выявлением ситуаций, в которых поведение программы является неправильным, нежелательным или не соответствующим спецификации. Для достижения указанной цели существует несколько видов тестирования [7].

1. Цель
   1. Функциональное тестирование направлено на проверку того, какие функции ПО реализованы, и того, насколько верно они реализованы.
   2. Нефункциональное – проверка корректности работы нефункциональных требований. Оценивается, КАК программный продукт работает. Эта проверка включает в себя следующие виды:
      1. **Тестирование производительности** – работа ПО под определённой нагрузкой.
      2. **Тестирование пользовательского интерфейса** – удобство пользователя при взаимодействии с разными параметрами интерфейса (кнопки, цвета, выравнивание и т. д.).
      3. **Тестирование UX** – правильность логики использования программного продукта.
      4. **Тестирование защищенности** – определение безопасности ПО: защищено ли оно от атак хакеров, несанкционированного доступа к данным и т. д.
      5. **Инсталляционное тестирование** – оценка вероятности возникновения проблем при установке, удалении, а также обновлении ПО.
      6. **Тестирование совместимости** – тестирование работы программного продукта в определённом окружении.
      7. **Тестирование надежности** – работа программы при длительной средней ожидаемой нагрузке.
      8. **Тестирование локализации** – оценка правильности версии программного продукта (языковой и культурный аспекты).
2. Степень автоматизации
   1. **Мануальное (ручное)** – без использования дополнительных программных средств, т. е. «вручную».
   2. **Автоматизированное** – с использованием программных средств (более детально в описании курса по автоматизации тестирования ПО).
3. Позитивность сценария
   1. **Позитивная проверка** – оценка ожидаемого поведения. Это тестирование проводится в первую очередь, ведь позволяет определить корректность работы программы.
   2. **Негативная** – определение устойчивости системы в нестандартной ситуации. Например, неожиданный сценарий взаимодействия пользователя с интерфейсом.
4. Доступ к коду программного продукта
   1. **Тестирование «белого ящика»** – с доступом к коду.
   2. **Тестирование «черного ящика»** – без доступа к коду продукта.
   3. **Тестирование «серого ящика»** – на основе ограниченного знания внутренней структуры ПО. В данном случае тестировщик не работает с кодом программного продукта, но он знаком с внутренней структурой программы и взаимодействием между компонентами.
5. Уровень
   1. **Модульное / юнит-тестирование** – проверка корректной работы отдельных единиц ПО, модулей. Этот вид тестирования могут выполнять сами разработчики.
   2. **Интеграционное тестирование** – проверка взаимодействия между несколькими единицами ПО.
   3. **Системное** – проверка работы приложения целиком.
   4. **Приёмочное** – оценка соответствия заявленным требованиям к программному продукту.
6. Исполнитель
   1. **Альфа-тестирование** – проверка программного продукта на поздней стадии разработки. Проводится разработчиками или тестировщиками.
   2. **Бета-тестирование** – оценка ПО перед выходом на рынок в фокус-группе или добровольцами. Отзывы собираются, анализируются и учитываются при внесении правок.
7. Формальность
   1. **Тестирование по тестам** – использование написанных заранее тест-кейсов.
   2. **Исследовательское тестирование** – одновременная разработка тестов и их использование.
   3. **Свободное тестирование** – проверка качества без разработки тестов и написания документации. Основывается на интуиции и опыте тестировщика.
8. Важность
   1. **Дымовое тестирование** – проверка самой важной функциональности программного продукта.
   2. **Тестирование критического пути** – проверка функциональности, используемой типичными пользователями в повседневной деятельности.
   3. **Расширенное тестирование** – проверка всей заявленной функциональности [7].

Для тестирования разрабатываемого приложения было выбрано функциональное тестирование, т.к. главной задачей является проверка на соответствие ПО функциональным требованиям. Так же этот вид тестирования можно провести самостоятельно, без лишних затрат вычислительных ресурсов. Кроме того, с его помощью можно проверить соответствие программы по сразу нескольким параметрам.

### Функциональное тестирование программы

Результаты функционального тестирования представлены в таблице 5.

Таблица 5 – План тестирования

|  |  |  |
| --- | --- | --- |
| **Состав теста** | **Ожидаемый результат** | **Наблюдаемый результат** |
| Поиск и сортировка  Размерность: 100 | Таблицы с характеристиками поиска и сортировки. Графики зависимости времени поиска и сортировки от размера массива | Таблицы с характеристиками поиска и сортировки. Графики зависимости времени поиска и сортировки от размера массива (рисунок 14) |
| Поиск и сортировка  Размерность: 10000 | Таблицы с характеристиками поиска и сортировки. Графики зависимости времени поиска и сортировки от размера массива | Таблицы с характеристиками поиска и сортировки. Графики зависимости времени поиска и сортировки от размера массива (рисунок 15) |
| Добавить в дерево значение 569 | Добавление в дерево значения 569. Отрисовка дерева | Добавление в дерево значения 569. Дерево отрисовалось (рисунок 16) |
| Добавить в дерево значения 7, 412, 253, 56, 752, 56, 745, 750 | Добавление в дерево значений 7, 412, 253, 56, 752, 56, 745, 750. Отрисовка дерева | Добавление в дерево значений 7, 412, 253, 56, 752, 56, 745, 750. Дерево отрисовалось (рисунок 17) |
| Удалить из дерева значение 745 | Удаление из дерева значения 745. Отрисовка дерева | Удаление из дерева значения 745. Дерево отрисовалось (рисунок 18) |
| Удалить из дерева значение 283 | Сообщение об отсутствии значения 283 | Сообщение об отсутствии значения 283 (рисунок 19) |
| Изменить значение 412 на 500 | Изменение значения 412 на 500. Отрисовка дерева | Изменение значения 412 на 500. Дерево отрисовалось (рисунок 20) |
| Изменить значение 413 на 500 | Сообщение об отсутствии значения 412 и невозможности операции | Сообщение об отсутствии значения 412 и невозможности операции (рисунок 21) |

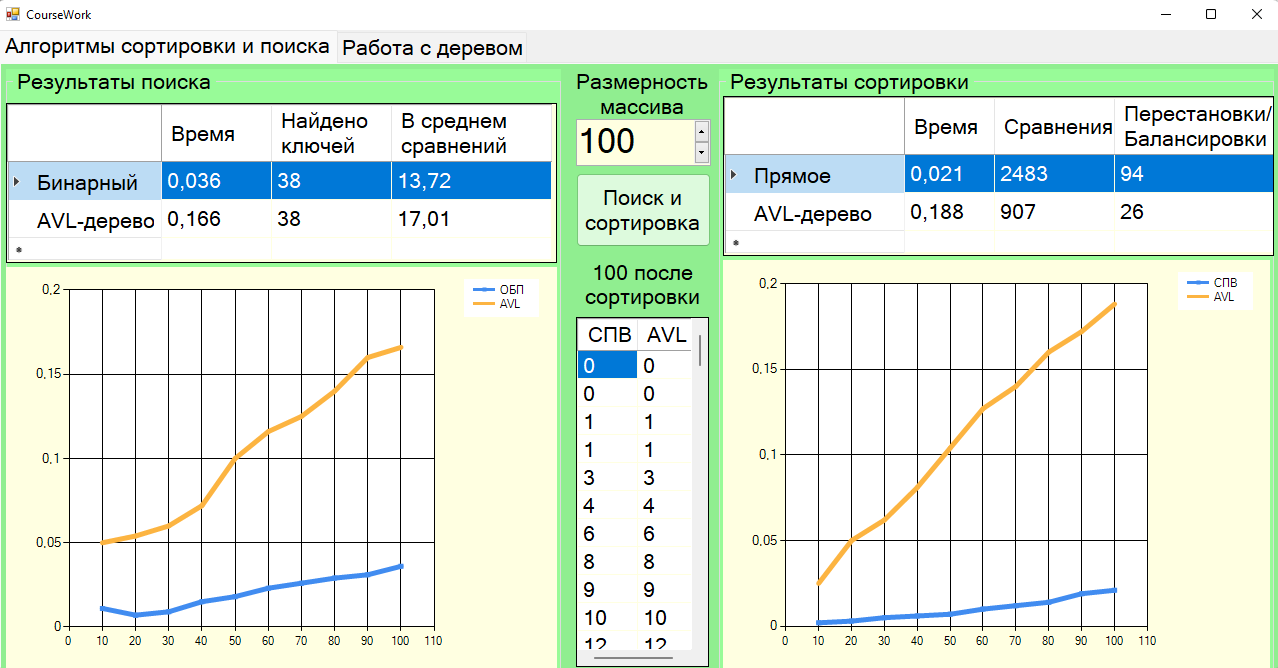


Рисунок 14 – Тест №1. Результаты поиска и сортировки.

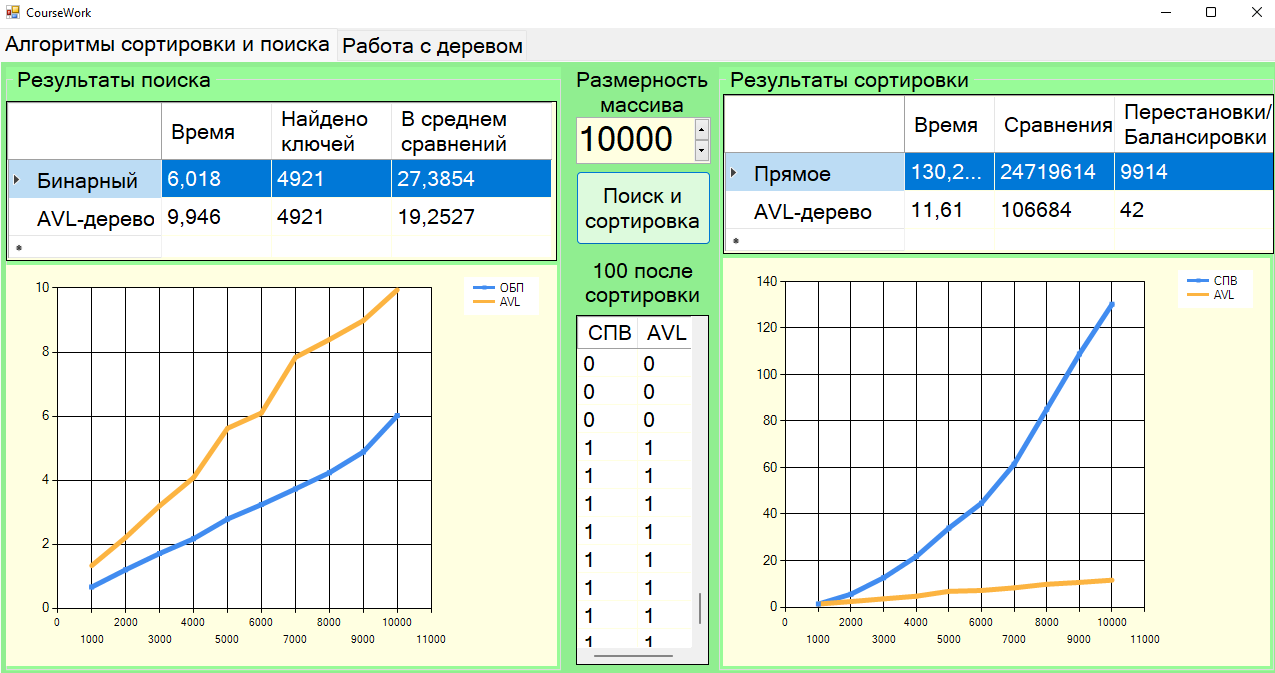


Рисунок 15 – Тест №2. Результаты поиска и сортировки.

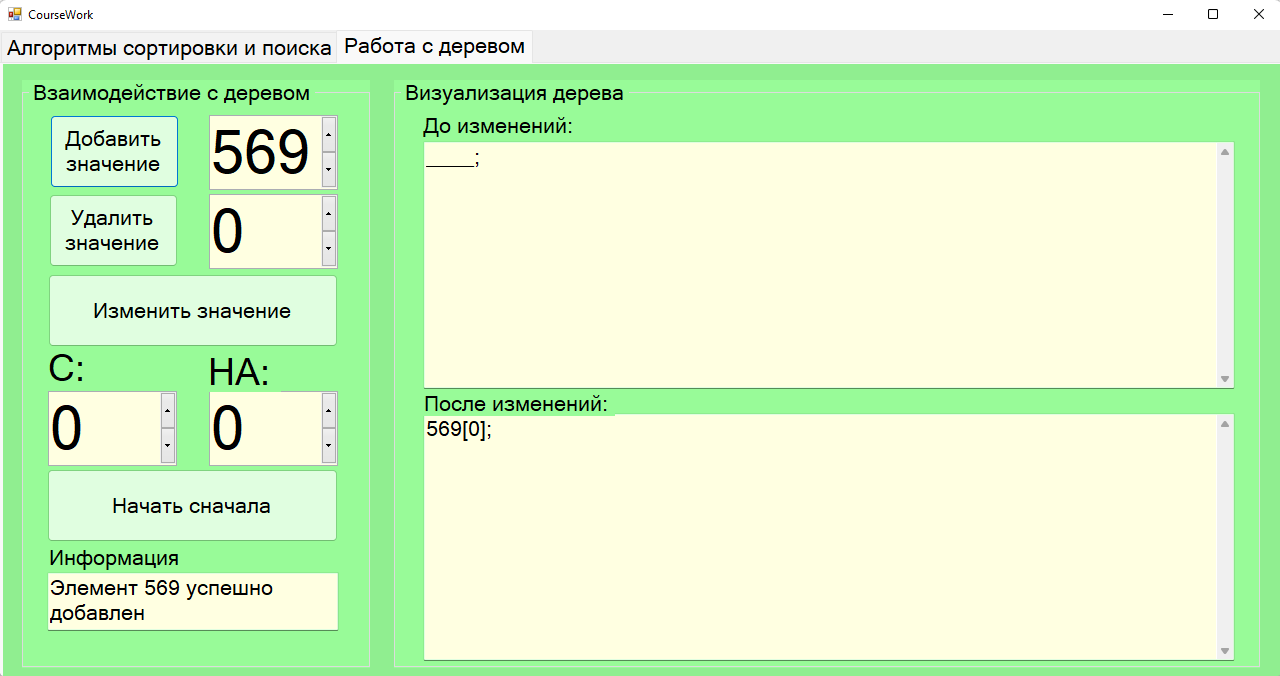


Рисунок 16 – Тест №3. Добавление значения 569 в АВЛ-дерево.

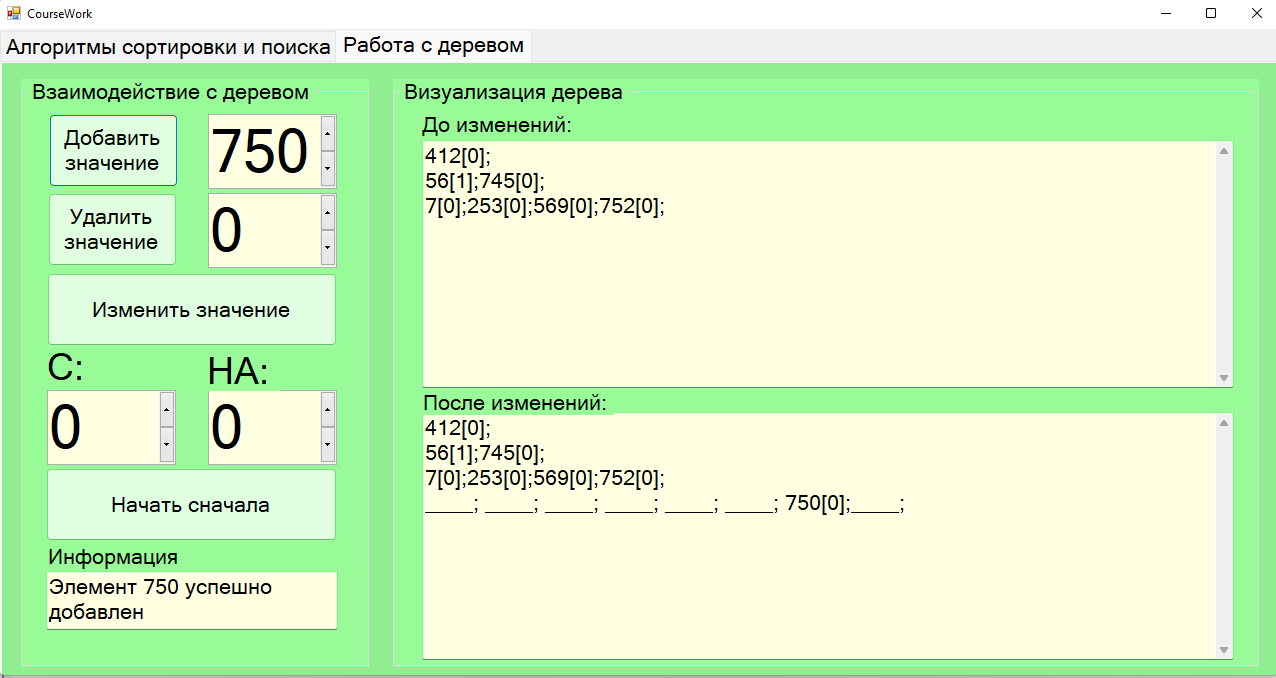


Рисунок 17 – Тест №4. Добавление значений в АВЛ-дерево.

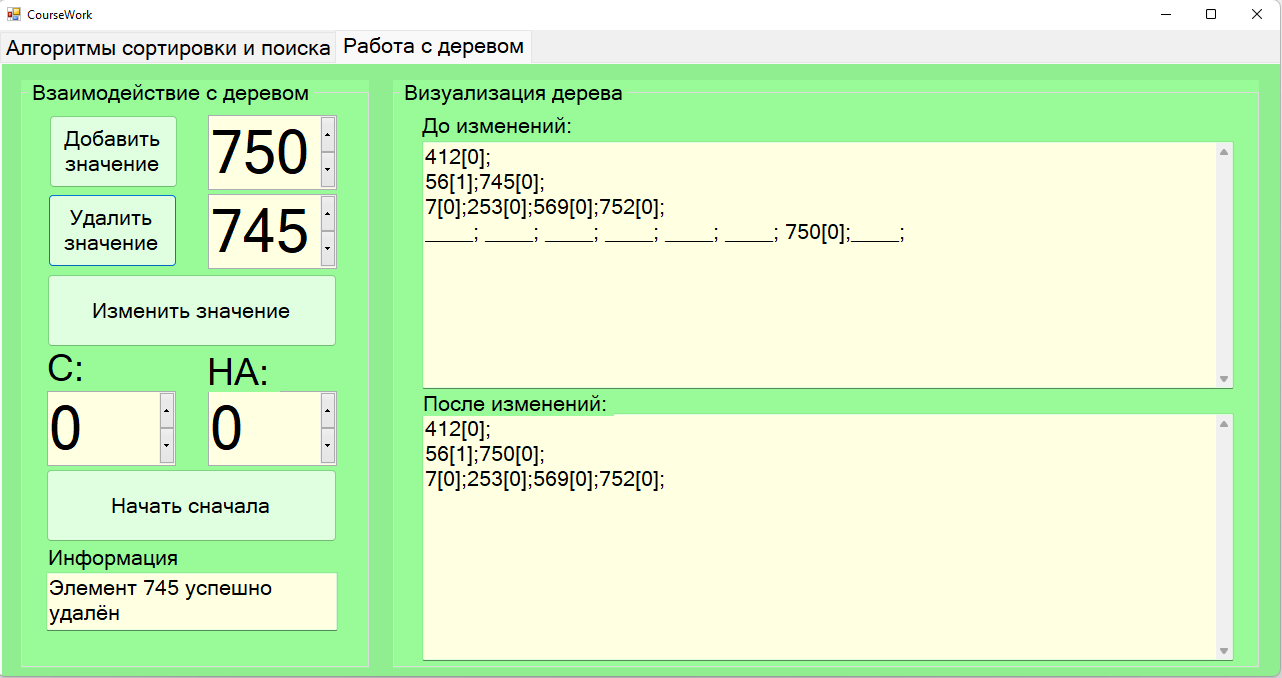


Рисунок 18 – Тест №5. Удаление значения 745 из AVL-дерева.

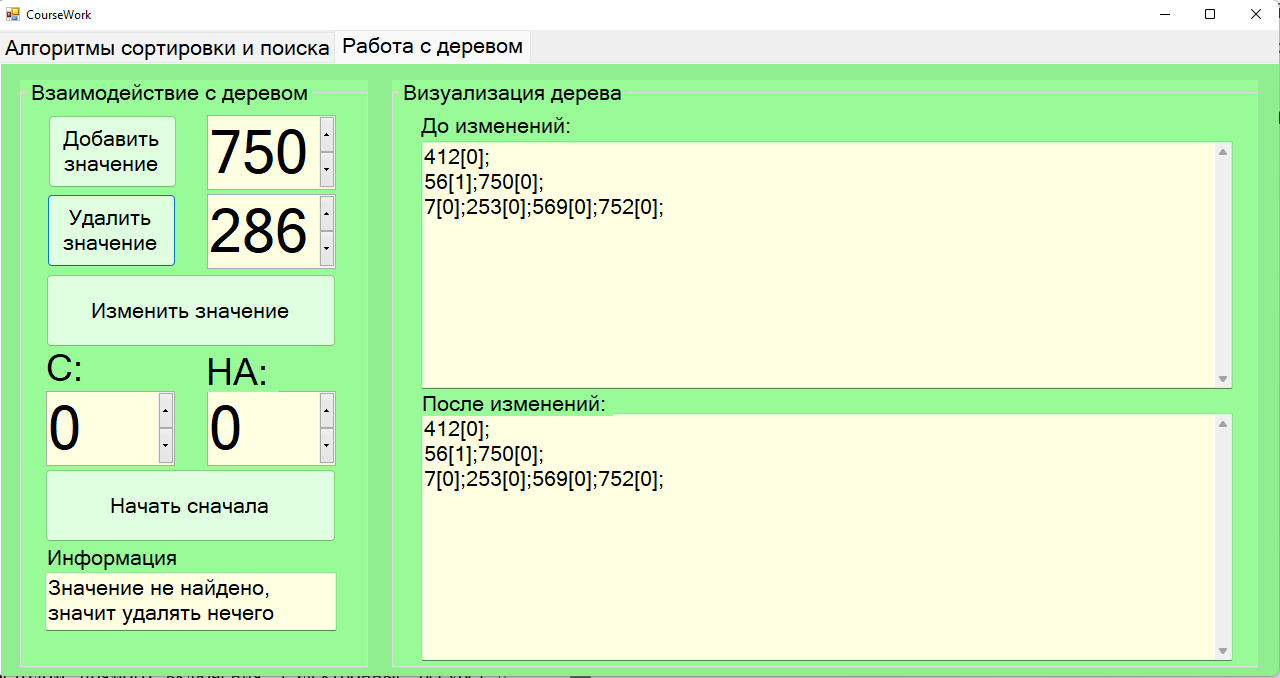


Рисунок 19 – Тест №6. Удаление значения 786 из АВЛ-дерева. Сообщение об ошибке из-за отсутствия элемента.

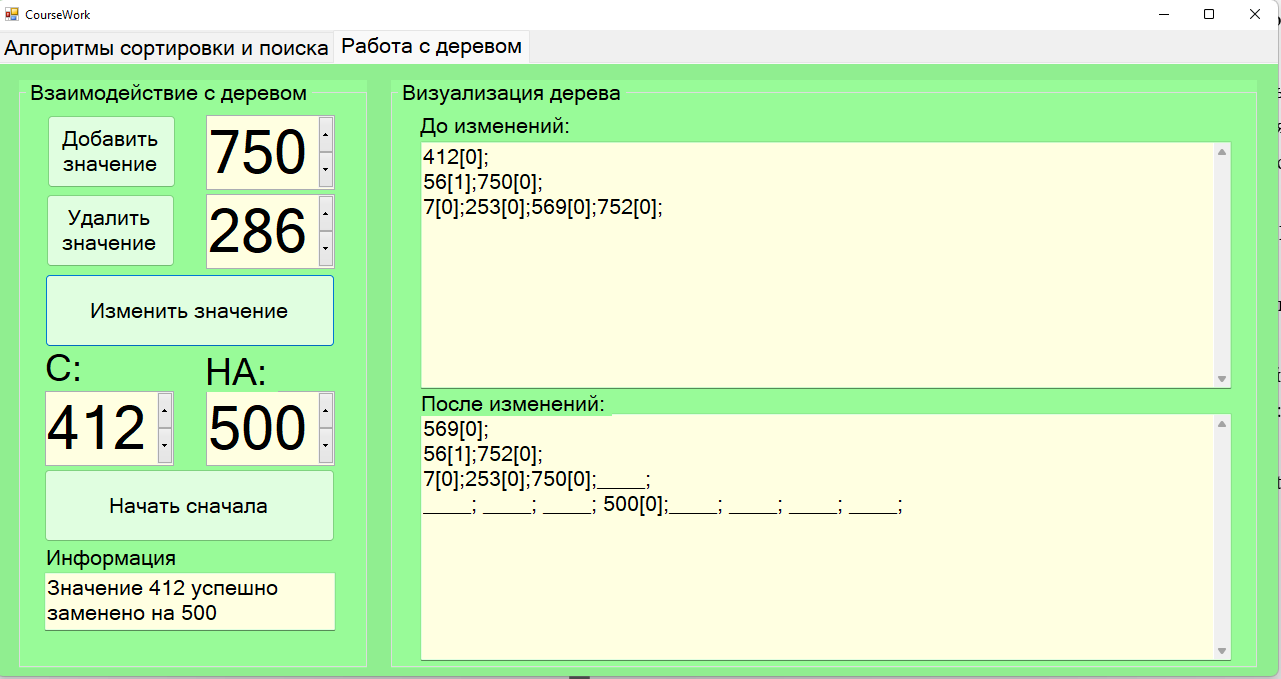


Рисунок 20 – Тест №7. Замена значения 412 на 500.

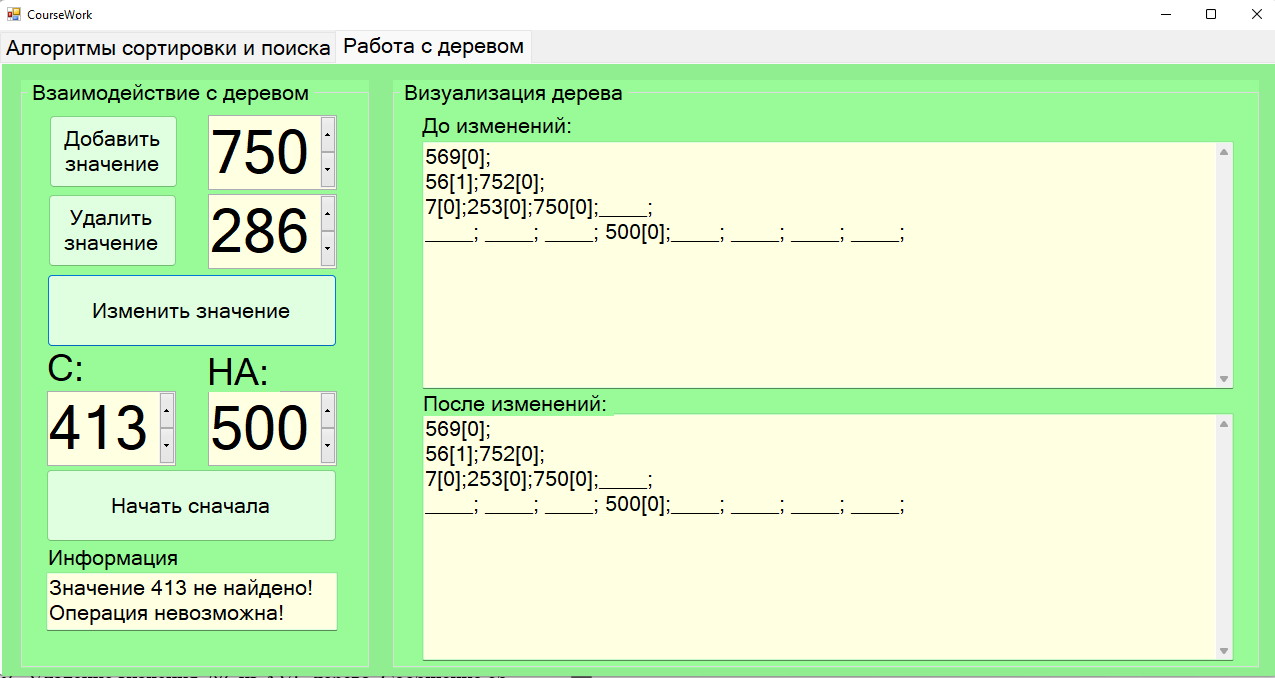


Рисунок 21 – Тест №8. Замена значения 413 на 500. Сообщение об ошибке из-за отсутствия элемента.

# Анализ результатов

Можно заметить, что при увеличении размера массива в 10 раз отношение времени сортировки прямым включением к времени сортировки АВЛ-деревом возрастает более чем в 9 раз. Таким образом, при малом числе элементов (до 200-300) эффективнее использовать сортировку прямым включением, при количестве элементов больше 1000 данный алгоритм менее эффективный, чем сортировка АВЛ-деревом. Это связано с тем, что при увеличении размера массива включение элемента в упорядоченную последовательность влечет за собой сдвиг все большей группы элементов, что более трудоемко, чем балансировка дерева. Для включения в массив i-того элемента в упорядоченную последовательность в худшем случае требует i сравнений и i сдвигов. Для добавления элемента в дерево потребуется сравнений и максимум 2 балансировки, что требует меньше вычислительной мощности и затрачивает меньше времени.



По результатам тестов получается, что время работы алгоритма оптимального бинарного поиска в среднем в 2,6 раз меньше, чем время поиска в АВЛ-дереве. Однако оба алгоритма в худшем случае делают сравнений, но переход к следующей итерации в АВЛ-дереве происходит медленнее вследствие того, что дерево – более сложная структура данных, нежели массив.

# Заключение

В ходе выполнения практической работы было разработано приложение «coursework.exe», реализующая алгоритмы работы сбалансированных деревьев. Был выполнен анализ алгоритмов сортировки и поиска в массиве и в дереве. Проверка кода была осуществлена по методу функционального тестирования. Результаты показывают корректную работу приложения.

В процессе разработки был получили опыт в создании проектов в среде Microsoft Visual Studio на языке С++.

# Список использованных источников

1. АВЛ-дерево: [Электронные ресурс] // Викиконспекты. https://neerc.ifmo.ru/wiki/index.php?title=АВЛ-дерево (дата обращения: 22.10.2021)
2. АВЛ-дерево: [Электронные ресурс] // Википедия – свободная энциклопедия. URL: https://ru.wikipedia.org/wiki/АВЛ-дерево (дата обращения: 22.10.2021)
3. Сортировка методом прямого включения: [Электронные ресурс] // Студопедия. URL: https://studopedia.ru/5\_116906\_sortirovka-metodom-pryamogo-vklyucheniya.html
4. Тестирование программного обеспечения: [Электронные ресурс] // Википедия – свободная энциклопедия. URL: https://ru.wikipedia.org/wiki/Тестирование\_программного\_обеспечения (дата обращения: 22.10.2021)
5. RAD (программирование): [Электронные ресурс] // Википедия – свободная энциклопедия. URL: https://ru.wikipedia.org/wiki/RAD\_(программирование) (дата обращения: 22.10.2021)
6. RAD модель: [Электронные ресурс] // Графика, дизайн, расчеты и практика программирования. URL: https://ru.education-wiki.com/5801526-rad-model (дата обращения: 22.10.2021)
7. Классификация видов тестирования: [Электронные ресурс] // Курсы тестирования ПО в Минске. URL: <https://qa-academy.by/qaacademy/news/klassifikaciya-vidov-testirovaniya/> (дата обращения: 06.11.2021)

# Приложение А. Листинг программы «coursework.exe»

**MyForm.h**

#pragma once

#include <chrono>

#include "AVL.h"

namespace coursework {

using namespace System;

using namespace System::ComponentModel;

using namespace Collections;

using namespace Windows::Forms;

using namespace Data;

using namespace Drawing;

public ref class MyForm : public Form {

public:

MyForm(void) {

N = 10000;

tree = new AVL();

tree\_demonstration = new AVL();

InitializeComponent();

numericUpDown1->Minimum = 0;

numericUpDown1->Maximum = 1000;

numericUpDown2->Minimum = 0;

numericUpDown2->Maximum = 1000;

numericUpDown3->Minimum = 0;

numericUpDown3->Maximum = 1000;

numericUpDown4->Minimum = 0;

numericUpDown4->Maximum = 1000;

numericUpDown5->Minimum = 100;

numericUpDown5->Maximum = 50000;

}

private:

AVL\* tree;

AVL\* tree\_demonstration;

int N;

bool search\_table\_flag = true;

Button^ button1, ^ button2, ^ button3, ^ button4, ^ button6;

DataGridView^ dataGridView1, ^ dataGridView2;

DataGridViewTextBoxColumn^ Column1, ^ Column2, ^ Column3;

GroupBox^ groupBox1, ^ groupBox2, ^ groupBox3, ^ groupBox4;

NumericUpDown^ numericUpDown1, ^ numericUpDown2, ^ numericUpDown3, ^ numericUpDown4, ^ numericUpDown5;

TabControl^ tabControl1;

TabPage^ tabPage1, ^ tabPage2;

TextBox^ textBox1, ^ textBox2, ^ textBox3;

Windows::Forms::Label^ label1;

Windows::Forms::Label^ label2;

Windows::Forms::Label^ label3;

Windows::Forms::Label^ label4;

Windows::Forms::Label^ label5;

Windows::Forms::Label^ label6;

DataVisualization::Charting::Chart^ chart2;

DataVisualization::Charting::Chart^ chart1;

private: System::Windows::Forms::Label^ label8;

private: System::Windows::Forms::Label^ label7;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Column4;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Column5;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Column6;

private: System::Windows::Forms::DataGridView^ dataGridView3;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Column7;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Column8;

// Обязательная переменная конструктора.

System::ComponentModel::Container^ components;

protected:

~MyForm() {

if (!components) return;

delete components;

}

// Требуемый метод для поддержки конструктора

void InitializeComponent(void);

static Void my\_form\_load(Object^ sender, EventArgs^ e);

// Добавление элемента

Void button1\_click(Object^ sender, EventArgs^ e);

// Удаление значения

Void button2\_click(Object^ sender, EventArgs^ e);

// Изменение значения

Void button3\_click(Object^ sender, EventArgs^ e);

// Поиск и сортировка

Void button4\_click(Object^ sender, EventArgs^ e);

// Начать сначала

Void button6\_click(Object^ sender, EventArgs^ e);

// Оптимальный бинарный поиск

static int opt\_bin\_looking(int arr[], const int n, const int key, double& comparisons);

// Сортировка прямым включением

static void forward\_include(int a[], const int n, int& permutations, int& comparisons);

static void forward\_include(int a[], const int n);

};

}

**MyForm.cpp**

#include "MyForm.h"

#include <Windows.h>

#include <algorithm>

using namespace coursework; // Название проекта

int WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int) {

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

Application::Run(gcnew MyForm);

return 0;

}

// Требуемый метод для поддержки конструктора

void MyForm::InitializeComponent() {

System::Windows::Forms::DataGridViewCellStyle^ dataGridViewCellStyle3 = (gcnew System::Windows::Forms::DataGridViewCellStyle());

System::Windows::Forms::DataVisualization::Charting::ChartArea^ chartArea5 = (gcnew System::Windows::Forms::DataVisualization::Charting::ChartArea());

System::Windows::Forms::DataVisualization::Charting::Legend^ legend5 = (gcnew System::Windows::Forms::DataVisualization::Charting::Legend());

System::Windows::Forms::DataVisualization::Charting::Series^ series9 = (gcnew System::Windows::Forms::DataVisualization::Charting::Series());

System::Windows::Forms::DataVisualization::Charting::Series^ series10 = (gcnew System::Windows::Forms::DataVisualization::Charting::Series());

System::Windows::Forms::DataVisualization::Charting::ChartArea^ chartArea6 = (gcnew System::Windows::Forms::DataVisualization::Charting::ChartArea());

System::Windows::Forms::DataVisualization::Charting::Legend^ legend6 = (gcnew System::Windows::Forms::DataVisualization::Charting::Legend());

System::Windows::Forms::DataVisualization::Charting::Series^ series11 = (gcnew System::Windows::Forms::DataVisualization::Charting::Series());

System::Windows::Forms::DataVisualization::Charting::Series^ series12 = (gcnew System::Windows::Forms::DataVisualization::Charting::Series());

this->tabControl1 = (gcnew System::Windows::Forms::TabControl());

this->tabPage1 = (gcnew System::Windows::Forms::TabPage());

this->label8 = (gcnew System::Windows::Forms::Label());

this->label7 = (gcnew System::Windows::Forms::Label());

this->dataGridView3 = (gcnew System::Windows::Forms::DataGridView());

this->button4 = (gcnew System::Windows::Forms::Button());

this->label3 = (gcnew System::Windows::Forms::Label());

this->numericUpDown5 = (gcnew System::Windows::Forms::NumericUpDown());

this->groupBox4 = (gcnew System::Windows::Forms::GroupBox());

this->dataGridView2 = (gcnew System::Windows::Forms::DataGridView());

this->Column4 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->Column5 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->Column6 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->chart2 = (gcnew System::Windows::Forms::DataVisualization::Charting::Chart());

this->groupBox3 = (gcnew System::Windows::Forms::GroupBox());

this->dataGridView1 = (gcnew System::Windows::Forms::DataGridView());

this->Column1 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->Column2 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->Column3 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->chart1 = (gcnew System::Windows::Forms::DataVisualization::Charting::Chart());

this->tabPage2 = (gcnew System::Windows::Forms::TabPage());

this->groupBox2 = (gcnew System::Windows::Forms::GroupBox());

this->label5 = (gcnew System::Windows::Forms::Label());

this->label4 = (gcnew System::Windows::Forms::Label());

this->textBox2 = (gcnew System::Windows::Forms::TextBox());

this->textBox1 = (gcnew System::Windows::Forms::TextBox());

this->groupBox1 = (gcnew System::Windows::Forms::GroupBox());

this->button6 = (gcnew System::Windows::Forms::Button());

this->label6 = (gcnew System::Windows::Forms::Label());

this->textBox3 = (gcnew System::Windows::Forms::TextBox());

this->label2 = (gcnew System::Windows::Forms::Label());

this->label1 = (gcnew System::Windows::Forms::Label());

this->numericUpDown4 = (gcnew System::Windows::Forms::NumericUpDown());

this->numericUpDown3 = (gcnew System::Windows::Forms::NumericUpDown());

this->numericUpDown2 = (gcnew System::Windows::Forms::NumericUpDown());

this->numericUpDown1 = (gcnew System::Windows::Forms::NumericUpDown());

this->button3 = (gcnew System::Windows::Forms::Button());

this->button2 = (gcnew System::Windows::Forms::Button());

this->button1 = (gcnew System::Windows::Forms::Button());

this->Column7 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->Column8 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->tabControl1->SuspendLayout();

this->tabPage1->SuspendLayout();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->dataGridView3))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->numericUpDown5))->BeginInit();

this->groupBox4->SuspendLayout();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->dataGridView2))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->chart2))->BeginInit();

this->groupBox3->SuspendLayout();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->dataGridView1))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->chart1))->BeginInit();

this->tabPage2->SuspendLayout();

this->groupBox2->SuspendLayout();

this->groupBox1->SuspendLayout();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->numericUpDown4))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->numericUpDown3))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->numericUpDown2))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->numericUpDown1))->BeginInit();

this->SuspendLayout();

//

// tabControl1

//

this->tabControl1->Controls->Add(this->tabPage1);

this->tabControl1->Controls->Add(this->tabPage2);

this->tabControl1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 15.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->tabControl1->Location = System::Drawing::Point(0, 0);

this->tabControl1->Margin = System::Windows::Forms::Padding(4, 5, 4, 5);

this->tabControl1->Name = L"tabControl1";

this->tabControl1->SelectedIndex = 0;

this->tabControl1->Size = System::Drawing::Size(1929, 1175);

this->tabControl1->TabIndex = 0;

//

// tabPage1

//

this->tabPage1->BackColor = System::Drawing::Color::LightGreen;

this->tabPage1->Controls->Add(this->label8);

this->tabPage1->Controls->Add(this->label7);

this->tabPage1->Controls->Add(this->dataGridView3);

this->tabPage1->Controls->Add(this->button4);

this->tabPage1->Controls->Add(this->label3);

this->tabPage1->Controls->Add(this->numericUpDown5);

this->tabPage1->Controls->Add(this->groupBox4);

this->tabPage1->Controls->Add(this->groupBox3);

this->tabPage1->Location = System::Drawing::Point(4, 46);

this->tabPage1->Margin = System::Windows::Forms::Padding(4, 5, 4, 5);

this->tabPage1->Name = L"tabPage1";

this->tabPage1->Padding = System::Windows::Forms::Padding(4, 5, 4, 5);

this->tabPage1->Size = System::Drawing::Size(1921, 1125);

this->tabPage1->TabIndex = 0;

this->tabPage1->Text = L"Алгоритмы сортировки и поиска";

//

// label8

//

this->label8->AutoSize = true;

this->label8->Location = System::Drawing::Point(868, 339);

this->label8->Name = L"label8";

this->label8->Size = System::Drawing::Size(182, 37);

this->label8->TabIndex = 6;

this->label8->Text = L"сортировки";

this->label8->TextAlign = System::Drawing::ContentAlignment::MiddleCenter;

//

// label7

//

this->label7->AutoSize = true;

this->label7->Location = System::Drawing::Point(879, 302);

this->label7->Name = L"label7";

this->label7->Size = System::Drawing::Size(160, 37);

this->label7->TabIndex = 5;

this->label7->Text = L"100 после";

this->label7->TextAlign = System::Drawing::ContentAlignment::MiddleCenter;

//

// dataGridView3

//

this->dataGridView3->BackgroundColor = System::Drawing::SystemColors::Info;

this->dataGridView3->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->dataGridView3->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(2) {

this->Column7,

this->Column8

});

this->dataGridView3->GridColor = System::Drawing::SystemColors::Info;

this->dataGridView3->Location = System::Drawing::Point(862, 390);

this->dataGridView3->Name = L"dataGridView3";

this->dataGridView3->RowHeadersVisible = false;

this->dataGridView3->RowHeadersWidthSizeMode = System::Windows::Forms::DataGridViewRowHeadersWidthSizeMode::AutoSizeToAllHeaders;

this->dataGridView3->RowTemplate->Height = 28;

this->dataGridView3->Size = System::Drawing::Size(199, 539);

this->dataGridView3->TabIndex = 4;

//

// button4

//

this->button4->Location = System::Drawing::Point(862, 167);

this->button4->Name = L"button4";

this->button4->Size = System::Drawing::Size(202, 114);

this->button4->TabIndex = 3;

this->button4->Text = L"Поиск и сортировка";

this->button4->UseVisualStyleBackColor = true;

this->button4->Click += gcnew System::EventHandler(this, &MyForm::button4\_click);

//

// label3

//

this->label3->AutoSize = true;

this->label3->Location = System::Drawing::Point(855, 8);

this->label3->Name = L"label3";

this->label3->Size = System::Drawing::Size(206, 74);

this->label3->TabIndex = 2;

this->label3->Text = L"Размерность\r\nмассива";

this->label3->TextAlign = System::Drawing::ContentAlignment::MiddleCenter;

//

// numericUpDown5

//

this->numericUpDown5->BackColor = System::Drawing::SystemColors::Info;

this->numericUpDown5->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 26.25F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->numericUpDown5->Location = System::Drawing::Point(862, 85);

this->numericUpDown5->Name = L"numericUpDown5";

this->numericUpDown5->Size = System::Drawing::Size(202, 67);

this->numericUpDown5->TabIndex = 1;

//

// groupBox4

//

this->groupBox4->BackColor = System::Drawing::Color::PaleGreen;

this->groupBox4->Controls->Add(this->dataGridView2);

this->groupBox4->Controls->Add(this->chart2);

this->groupBox4->Location = System::Drawing::Point(1077, 8);

this->groupBox4->Name = L"groupBox4";

this->groupBox4->Size = System::Drawing::Size(832, 930);

this->groupBox4->TabIndex = 0;

this->groupBox4->TabStop = false;

this->groupBox4->Text = L"Результаты сортировки";

//

// dataGridView2

//

this->dataGridView2->BackgroundColor = System::Drawing::SystemColors::Info;

dataGridViewCellStyle3->Alignment = System::Windows::Forms::DataGridViewContentAlignment::MiddleLeft;

dataGridViewCellStyle3->BackColor = System::Drawing::SystemColors::Info;

dataGridViewCellStyle3->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 15.75F, System::Drawing::FontStyle::Regular,

System::Drawing::GraphicsUnit::Point, static\_cast<System::Byte>(204)));

dataGridViewCellStyle3->ForeColor = System::Drawing::SystemColors::WindowText;

dataGridViewCellStyle3->SelectionBackColor = System::Drawing::SystemColors::Highlight;

dataGridViewCellStyle3->SelectionForeColor = System::Drawing::SystemColors::Info;

dataGridViewCellStyle3->WrapMode = System::Windows::Forms::DataGridViewTriState::True;

this->dataGridView2->ColumnHeadersDefaultCellStyle = dataGridViewCellStyle3;

this->dataGridView2->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->dataGridView2->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(3) {

this->Column4,

this->Column5, this->Column6

});

this->dataGridView2->GridColor = System::Drawing::SystemColors::Info;

this->dataGridView2->Location = System::Drawing::Point(6, 42);

this->dataGridView2->Name = L"dataGridView2";

this->dataGridView2->RowHeadersBorderStyle = System::Windows::Forms::DataGridViewHeaderBorderStyle::Sunken;

this->dataGridView2->RowHeadersWidthSizeMode = System::Windows::Forms::DataGridViewRowHeadersWidthSizeMode::AutoSizeToAllHeaders;

this->dataGridView2->RowTemplate->Height = 38;

this->dataGridView2->ScrollBars = System::Windows::Forms::ScrollBars::None;

this->dataGridView2->SelectionMode = System::Windows::Forms::DataGridViewSelectionMode::FullRowSelect;

this->dataGridView2->Size = System::Drawing::Size(826, 246);

this->dataGridView2->TabIndex = 2;

//

// Column4

//

this->Column4->HeaderText = L"Время";

this->Column4->MinimumWidth = 8;

this->Column4->Name = L"Column4";

this->Column4->Width = 90;

//

// Column5

//

this->Column5->HeaderText = L"Сравнения";

this->Column5->MinimumWidth = 8;

this->Column5->Name = L"Column5";

this->Column5->Width = 120;

//

// Column6

//

this->Column6->HeaderText = L"Перестановки/\nБалансировки";

this->Column6->MinimumWidth = 8;

this->Column6->Name = L"Column6";

this->Column6->Width = 328;

//

// chart2

//

this->chart2->BackColor = System::Drawing::SystemColors::Info;

this->chart2->BorderlineColor = System::Drawing::SystemColors::Info;

chartArea5->Name = L"ChartArea1";

this->chart2->ChartAreas->Add(chartArea5);

legend5->Name = L"Legend1";

this->chart2->Legends->Add(legend5);

this->chart2->Location = System::Drawing::Point(6, 294);

this->chart2->Name = L"chart2";

series9->BorderWidth = 5;

series9->ChartArea = L"ChartArea1";

series9->ChartType = System::Windows::Forms::DataVisualization::Charting::SeriesChartType::Line;

series9->Legend = L"Legend1";

series9->LegendText = L"СПВ";

series9->MarkerStyle = System::Windows::Forms::DataVisualization::Charting::MarkerStyle::Square;

series9->Name = L"Series1";

series10->BorderWidth = 5;

series10->ChartArea = L"ChartArea1";

series10->ChartType = System::Windows::Forms::DataVisualization::Charting::SeriesChartType::Line;

series10->Legend = L"Legend1";

series10->LegendText = L"AVL ";

series10->Name = L"Series2";

this->chart2->Series->Add(series9);

this->chart2->Series->Add(series10);

this->chart2->Size = System::Drawing::Size(820, 627);

this->chart2->TabIndex = 0;

this->chart2->Text = L"chart2";

//

// groupBox3

//

this->groupBox3->BackColor = System::Drawing::Color::PaleGreen;

this->groupBox3->Controls->Add(this->dataGridView1);

this->groupBox3->Controls->Add(this->chart1);

this->groupBox3->Location = System::Drawing::Point(8, 8);

this->groupBox3->Name = L"groupBox3";

this->groupBox3->Size = System::Drawing::Size(832, 930);

this->groupBox3->TabIndex = 0;

this->groupBox3->TabStop = false;

this->groupBox3->Text = L"Результаты поиска";

//

// dataGridView1

//

this->dataGridView1->BackgroundColor = System::Drawing::SystemColors::Info;

this->dataGridView1->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->dataGridView1->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(3) {

this->Column1,

this->Column2, this->Column3

});

this->dataGridView1->GridColor = System::Drawing::SystemColors::Info;

this->dataGridView1->Location = System::Drawing::Point(0, 52);

this->dataGridView1->Name = L"dataGridView1";

this->dataGridView1->RowHeadersBorderStyle = System::Windows::Forms::DataGridViewHeaderBorderStyle::Sunken;

this->dataGridView1->RowHeadersWidth = 155;

this->dataGridView1->RowTemplate->Height = 38;

this->dataGridView1->ScrollBars = System::Windows::Forms::ScrollBars::None;

this->dataGridView1->Size = System::Drawing::Size(826, 246);

this->dataGridView1->TabIndex = 1;

//

// Column1

//

this->Column1->HeaderText = L"Время";

this->Column1->MinimumWidth = 8;

this->Column1->Name = L"Column1";

this->Column1->Width = 110;

//

// Column2

//

this->Column2->HeaderText = L"Найдено ключей";

this->Column2->MinimumWidth = 8;

this->Column2->Name = L"Column2";

this->Column2->Width = 120;

//

// Column3

//

this->Column3->HeaderText = L"В среднем сравнений";

this->Column3->MinimumWidth = 8;

this->Column3->Name = L"Column3";

this->Column3->Width = 160;

//

// chart1

//

this->chart1->BackColor = System::Drawing::SystemColors::Info;

this->chart1->BorderlineColor = System::Drawing::SystemColors::Info;

chartArea6->Name = L"ChartArea1";

this->chart1->ChartAreas->Add(chartArea6);

legend6->Name = L"Legend1";

this->chart1->Legends->Add(legend6);

this->chart1->Location = System::Drawing::Point(0, 304);

this->chart1->Name = L"chart1";

series11->BorderWidth = 5;

series11->ChartArea = L"ChartArea1";

series11->ChartType = System::Windows::Forms::DataVisualization::Charting::SeriesChartType::Line;

series11->Legend = L"Legend1";

series11->LegendText = L"ОБП";

series11->MarkerStyle = System::Windows::Forms::DataVisualization::Charting::MarkerStyle::Square;

series11->Name = L"Series1";

series12->BorderWidth = 5;

series12->ChartArea = L"ChartArea1";

series12->ChartType = System::Windows::Forms::DataVisualization::Charting::SeriesChartType::Line;

series12->Legend = L"Legend1";

series12->LegendText = L"AVL";

series12->Name = L"Series2";

this->chart1->Series->Add(series11);

this->chart1->Series->Add(series12);

this->chart1->Size = System::Drawing::Size(826, 617);

this->chart1->TabIndex = 0;

this->chart1->Text = L"chart1";

//

// tabPage2

//

this->tabPage2->BackColor = System::Drawing::Color::LightGreen;

this->tabPage2->Controls->Add(this->groupBox2);

this->tabPage2->Controls->Add(this->groupBox1);

this->tabPage2->Location = System::Drawing::Point(4, 46);

this->tabPage2->Margin = System::Windows::Forms::Padding(4, 5, 4, 5);

this->tabPage2->Name = L"tabPage2";

this->tabPage2->Padding = System::Windows::Forms::Padding(4, 5, 4, 5);

this->tabPage2->Size = System::Drawing::Size(1921, 1125);

this->tabPage2->TabIndex = 1;

this->tabPage2->Text = L"Работа с деревом";

//

// groupBox2

//

this->groupBox2->BackColor = System::Drawing::Color::PaleGreen;

this->groupBox2->Controls->Add(this->label5);

this->groupBox2->Controls->Add(this->label4);

this->groupBox2->Controls->Add(this->textBox2);

this->groupBox2->Controls->Add(this->textBox1);

this->groupBox2->Location = System::Drawing::Point(586, 25);

this->groupBox2->Margin = System::Windows::Forms::Padding(4, 5, 4, 5);

this->groupBox2->Name = L"groupBox2";

this->groupBox2->Padding = System::Windows::Forms::Padding(4, 5, 4, 5);

this->groupBox2->Size = System::Drawing::Size(1299, 905);

this->groupBox2->TabIndex = 1;

this->groupBox2->TabStop = false;

this->groupBox2->Text = L"Визуализация дерева";

//

// label5

//

this->label5->AutoSize = true;

this->label5->Location = System::Drawing::Point(37, 479);

this->label5->Margin = System::Windows::Forms::Padding(4, 0, 4, 0);

this->label5->Name = L"label5";

this->label5->Size = System::Drawing::Size(273, 37);

this->label5->TabIndex = 3;

this->label5->Text = L"После изменений:";

//

// label4

//

this->label4->AutoSize = true;

this->label4->Location = System::Drawing::Point(36, 51);

this->label4->Margin = System::Windows::Forms::Padding(4, 0, 4, 0);

this->label4->Name = L"label4";

this->label4->Size = System::Drawing::Size(225, 37);

this->label4->TabIndex = 2;

this->label4->Text = L"До изменений:";

//

// textBox2

//

this->textBox2->BackColor = System::Drawing::SystemColors::Info;

this->textBox2->Location = System::Drawing::Point(43, 512);

this->textBox2->Margin = System::Windows::Forms::Padding(4, 5, 4, 5);

this->textBox2->Multiline = true;

this->textBox2->Name = L"textBox2";

this->textBox2->ScrollBars = System::Windows::Forms::ScrollBars::Both;

this->textBox2->Size = System::Drawing::Size(1216, 380);

this->textBox2->TabIndex = 1;

//

// textBox1

//

this->textBox1->BackColor = System::Drawing::SystemColors::Info;

this->textBox1->Location = System::Drawing::Point(44, 94);

this->textBox1->Margin = System::Windows::Forms::Padding(4, 5, 4, 5);

this->textBox1->Multiline = true;

this->textBox1->Name = L"textBox1";

this->textBox1->ScrollBars = System::Windows::Forms::ScrollBars::Both;

this->textBox1->Size = System::Drawing::Size(1216, 380);

this->textBox1->TabIndex = 0;

//

// groupBox1

//

this->groupBox1->BackColor = System::Drawing::Color::PaleGreen;

this->groupBox1->Controls->Add(this->button6);

this->groupBox1->Controls->Add(this->label6);

this->groupBox1->Controls->Add(this->textBox3);

this->groupBox1->Controls->Add(this->label2);

this->groupBox1->Controls->Add(this->label1);

this->groupBox1->Controls->Add(this->numericUpDown4);

this->groupBox1->Controls->Add(this->numericUpDown3);

this->groupBox1->Controls->Add(this->numericUpDown2);

this->groupBox1->Controls->Add(this->numericUpDown1);

this->groupBox1->Controls->Add(this->button3);

this->groupBox1->Controls->Add(this->button2);

this->groupBox1->Controls->Add(this->button1);

this->groupBox1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 15.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->groupBox1->Location = System::Drawing::Point(28, 25);

this->groupBox1->Margin = System::Windows::Forms::Padding(4, 5, 4, 5);

this->groupBox1->Name = L"groupBox1";

this->groupBox1->Padding = System::Windows::Forms::Padding(4, 5, 4, 5);

this->groupBox1->Size = System::Drawing::Size(522, 905);

this->groupBox1->TabIndex = 0;

this->groupBox1->TabStop = false;

this->groupBox1->Text = L"Взаимодействие с деревом";

//

// button6

//

this->button6->Location = System::Drawing::Point(38, 598);

this->button6->Margin = System::Windows::Forms::Padding(4, 5, 4, 5);

this->button6->Name = L"button6";

this->button6->Size = System::Drawing::Size(436, 112);

this->button6->TabIndex = 12;

this->button6->Text = L"Начать сначала";

this->button6->UseVisualStyleBackColor = true;

this->button6->Click += gcnew System::EventHandler(this, &MyForm::button6\_click);

//

// label6

//

this->label6->AutoSize = true;

this->label6->Location = System::Drawing::Point(33, 715);

this->label6->Margin = System::Windows::Forms::Padding(4, 0, 4, 0);

this->label6->Name = L"label6";

this->label6->Size = System::Drawing::Size(200, 37);

this->label6->TabIndex = 11;

this->label6->Text = L"Информация";

//

// textBox3

//

this->textBox3->BackColor = System::Drawing::SystemColors::Info;

this->textBox3->Location = System::Drawing::Point(38, 757);

this->textBox3->Margin = System::Windows::Forms::Padding(4, 5, 4, 5);

this->textBox3->Multiline = true;

this->textBox3->Name = L"textBox3";

this->textBox3->Size = System::Drawing::Size(436, 89);

this->textBox3->TabIndex = 10;

this->textBox3->Text = L"[n] - количество повторов";

//

// label2

//

this->label2->AutoSize = true;

this->label2->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 27.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label2->Location = System::Drawing::Point(269, 415);

this->label2->Margin = System::Windows::Forms::Padding(4, 0, 4, 0);

this->label2->Name = L"label2";

this->label2->Size = System::Drawing::Size(121, 64);

this->label2->TabIndex = 8;

this->label2->Text = L"НА:";

//

// label1

//

this->label1->AutoSize = true;

this->label1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 27.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label1->Location = System::Drawing::Point(28, 410);

this->label1->Margin = System::Windows::Forms::Padding(4, 0, 4, 0);

this->label1->Name = L"label1";

this->label1->Size = System::Drawing::Size(84, 64);

this->label1->TabIndex = 7;

this->label1->Text = L"С:";

//

// numericUpDown4

//

this->numericUpDown4->BackColor = System::Drawing::SystemColors::Info;

this->numericUpDown4->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 45, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->numericUpDown4->Location = System::Drawing::Point(280, 479);

this->numericUpDown4->Margin = System::Windows::Forms::Padding(4, 5, 4, 5);

this->numericUpDown4->Name = L"numericUpDown4";

this->numericUpDown4->Size = System::Drawing::Size(194, 109);

this->numericUpDown4->TabIndex = 6;

//

// numericUpDown3

//

this->numericUpDown3->BackColor = System::Drawing::SystemColors::Info;

this->numericUpDown3->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 45, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->numericUpDown3->Location = System::Drawing::Point(39, 479);

this->numericUpDown3->Margin = System::Windows::Forms::Padding(4, 5, 4, 5);

this->numericUpDown3->Name = L"numericUpDown3";

this->numericUpDown3->Size = System::Drawing::Size(194, 109);

this->numericUpDown3->TabIndex = 5;

//

// numericUpDown2

//

this->numericUpDown2->BackColor = System::Drawing::SystemColors::Info;

this->numericUpDown2->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 45, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->numericUpDown2->Location = System::Drawing::Point(280, 176);

this->numericUpDown2->Margin = System::Windows::Forms::Padding(4, 5, 4, 5);

this->numericUpDown2->Name = L"numericUpDown2";

this->numericUpDown2->Size = System::Drawing::Size(194, 109);

this->numericUpDown2->TabIndex = 4;

//

// numericUpDown1

//

this->numericUpDown1->BackColor = System::Drawing::SystemColors::Info;

this->numericUpDown1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 45, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->numericUpDown1->Location = System::Drawing::Point(280, 54);

this->numericUpDown1->Margin = System::Windows::Forms::Padding(4, 5, 4, 5);

this->numericUpDown1->Name = L"numericUpDown1";

this->numericUpDown1->Size = System::Drawing::Size(194, 109);

this->numericUpDown1->TabIndex = 3;

//

// button3

//

this->button3->Location = System::Drawing::Point(39, 298);

this->button3->Margin = System::Windows::Forms::Padding(4, 5, 4, 5);

this->button3->Name = L"button3";

this->button3->Size = System::Drawing::Size(435, 112);

this->button3->TabIndex = 2;

this->button3->Text = L"Изменить значение";

this->button3->UseVisualStyleBackColor = true;

this->button3->Click += gcnew System::EventHandler(this, &MyForm::button3\_click);

//

// button2

//

this->button2->Location = System::Drawing::Point(40, 176);

this->button2->Margin = System::Windows::Forms::Padding(4, 5, 4, 5);

this->button2->Name = L"button2";

this->button2->Size = System::Drawing::Size(194, 112);

this->button2->TabIndex = 1;

this->button2->Text = L"Удалить значение";

this->button2->UseVisualStyleBackColor = true;

this->button2->Click += gcnew System::EventHandler(this, &MyForm::button2\_click);

//

// button1

//

this->button1->Location = System::Drawing::Point(42, 54);

this->button1->Margin = System::Windows::Forms::Padding(4, 5, 4, 5);

this->button1->Name = L"button1";

this->button1->Size = System::Drawing::Size(194, 112);

this->button1->TabIndex = 0;

this->button1->Text = L"Добавить значение";

this->button1->UseVisualStyleBackColor = true;

this->button1->Click += gcnew System::EventHandler(this, &MyForm::button1\_click);

//

// Column7

//

this->Column7->HeaderText = L"СПВ";

this->Column7->MinimumWidth = 8;

this->Column7->Name = L"Column7";

this->Column7->Width = 60;

//

// Column8

//

this->Column8->HeaderText = L"AVL";

this->Column8->MinimumWidth = 8;

this->Column8->Name = L"Column8";

this->Column8->Width = 54;

//

// MyForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(9, 20);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->ClientSize = System::Drawing::Size(1924, 983);

this->Controls->Add(this->tabControl1);

this->Margin = System::Windows::Forms::Padding(4, 5, 4, 5);

this->Name = L"MyForm";

this->Text = L"CourseWork";

this->tabControl1->ResumeLayout(false);

this->tabPage1->ResumeLayout(false);

this->tabPage1->PerformLayout();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->dataGridView3))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->numericUpDown5))->EndInit();

this->groupBox4->ResumeLayout(false);

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->dataGridView2))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->chart2))->EndInit();

this->groupBox3->ResumeLayout(false);

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->dataGridView1))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->chart1))->EndInit();

this->tabPage2->ResumeLayout(false);

this->groupBox2->ResumeLayout(false);

this->groupBox2->PerformLayout();

this->groupBox1->ResumeLayout(false);

this->groupBox1->PerformLayout();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->numericUpDown4))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->numericUpDown3))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->numericUpDown2))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->numericUpDown1))->EndInit();

this->ResumeLayout(false);

}

void MyForm::my\_form\_load(Object^ sender, EventArgs^ e) { }

// Добавление элемента

void MyForm::button1\_click(Object^ sender, EventArgs^ e) {

tree\_demonstration->print\_tree(textBox1);

const int add\_value = Convert::ToInt32(numericUpDown1->Value);

const string message\_add = "Элемент " + to\_string(add\_value) + " успешно добавлен";

tree\_demonstration->add(add\_value);

tree\_demonstration->print\_tree(textBox2);

textBox3->Text = gcnew String(message\_add.c\_str());

}

// Удаление значения

void MyForm::button2\_click(Object^ sender, EventArgs^ e) {

string message\_remove;

if (tree\_demonstration == nullptr) {

message\_remove = "Операция невозможна! Дерево пустое";

textBox3->Text = gcnew String(message\_remove.c\_str());

return;

}

else {

tree\_demonstration->print\_tree(textBox1);

const int remove\_value = Convert::ToInt32(numericUpDown2->Value);

message\_remove = "Элемент " + to\_string(remove\_value) + " успешно удалён";

tree\_demonstration->remove(remove\_value, message\_remove);

tree\_demonstration->print\_tree(textBox2);

textBox3->Text = gcnew String(message\_remove.c\_str());

}

}

// Изменение значения

void MyForm::button3\_click(Object^ sender, EventArgs^ e) {

string message\_change;

if (tree\_demonstration == nullptr) {

message\_change = "Операция невозможна! Дерево пустое";

textBox3->Text = gcnew String(message\_change.c\_str());

return;

}

else {

tree\_demonstration->print\_tree(textBox1);

const int last = Convert::ToInt32(numericUpDown3->Value);

const int future = Convert::ToInt32(numericUpDown4->Value);

tree\_demonstration->remove(last, message\_change);

if (message\_change != "Значение не найдено, значит удалять нечего") {

tree\_demonstration->add(future);

message\_change = "Значение " + to\_string(last) + " успешно заменено на " + to\_string(future);

}

else

message\_change = "Значение " + to\_string(last) + " не найдено! Операция невозможна!";

tree\_demonstration->print\_tree(textBox2);

textBox3->Text = gcnew String(message\_change.c\_str());

}

}

// Поиск и сортировка

void MyForm::button4\_click(Object^ sender, EventArgs^ e) {

// Генерация строк таблица, если ещё не произошла

if (search\_table\_flag) {

tree->set\_N(N);

dataGridView1->Rows->Add();

dataGridView1->Rows[0]->HeaderCell->Value = "Бинарный";

dataGridView1->Rows[0]->HeaderCell->Size.Width = 240;

dataGridView1->Rows->Add();

dataGridView1->Rows[1]->HeaderCell->Value = "AVL-дерево";

dataGridView1->Rows[1]->HeaderCell->Size.Width = 240;

dataGridView2->Rows->Add();

dataGridView2->Rows[0]->HeaderCell->Value = "Прямое";

dataGridView2->Rows[0]->HeaderCell->Size.Width = 240;

dataGridView2->Rows->Add();

dataGridView2->Rows[1]->HeaderCell->Value = "AVL-дерево";

dataGridView2->Rows[1]->HeaderCell->Size.Width = 240;

for (int i = 0; i < 100; ++i)

dataGridView3->Rows->Add();

search\_table\_flag = false;

}

N = Convert::ToInt32(numericUpDown5->Value);

int\* arr\_values = new int[N]; // Массив значений для поиск

int\* arr\_sorting = new int[N]; // Массив значений для сортировки

int\* arr\_keys = new int[N]; // Массив ключей для поиска

for (int i = 0; i < N; ++i) {

arr\_values[i] = rand() % 100;

arr\_sorting[i] = arr\_values[i];

arr\_keys[i] = rand() % 200;

}

#pragma region Search

chart1->Series["Series1"]->Points->Clear(); // Очистка графиков

chart1->Series["Series2"]->Points->Clear();

for (int i = N / 10; i <= N; i = i + N / 10) {

// Поиск в AVL-дереве

auto begin\_search\_tree\_chart = chrono::high\_resolution\_clock::now(); // Время начала работы алгоритма

tree = new AVL(i); // Создание нового дерева

for (int j = 0; j < i; ++j) // Заполнение дерева

tree->add(arr\_values[j]);

double comparison\_tree\_search\_chart = 0; // Количество сравнений

int found\_quantity\_tree\_chart = 0; // Количество найденных ключей

for (int j = 0; j < i; ++j) // Поиск ключей в дереве

found\_quantity\_tree\_chart += tree->avl\_search(arr\_keys[j], comparison\_tree\_search\_chart);

auto end\_search\_tree\_chart = chrono::high\_resolution\_clock::now(); // Время завершения работы алгоритма

auto time\_search\_tree\_chart = chrono::duration\_cast<chrono::duration < double>>(end\_search\_tree\_chart - begin\_search\_tree\_chart);

chart1->Series["Series2"]->Points->AddXY(i, time\_search\_tree\_chart.count() \* 10000);

// Бинарый поиск

auto begin\_binary\_search\_chart = chrono::high\_resolution\_clock::now(); // Время начала работы алгоритма

sort(arr\_values, arr\_values + i); // Сортировка массива для бинарного поиска (сортировка прямым включением примерно в 3 раза дольше)

double comparison\_binary\_search\_chart = 0; // Количество сравнений

int found\_quantity\_binary\_chart = 0; // Количество найденных элементов

for (int j = 0; j < i; ++j) // Бианарный поиск

found\_quantity\_binary\_chart += opt\_bin\_looking(arr\_values, i, arr\_keys[j], comparison\_binary\_search\_chart);

auto end\_binary\_search\_chart = chrono::high\_resolution\_clock::now(); // Время завершения работы алгоритма

auto time\_binary\_search\_chart = chrono::duration\_cast<chrono::duration < double>>(end\_binary\_search\_chart - begin\_binary\_search\_chart);

chart1->Series["Series1"]->Points->AddXY(i, time\_binary\_search\_chart.count() \* 10000);

if (i == (N / 10) \* 10) {

// Заполнение таблицы с характеристиками поиска

dataGridView1->Rows[0]->Cells[0]->Value = Convert::ToString(time\_binary\_search\_chart.count() \* 10000);

dataGridView1->Rows[0]->Cells[1]->Value = found\_quantity\_binary\_chart;

dataGridView1->Rows[0]->Cells[2]->Value = comparison\_binary\_search\_chart / N;

dataGridView1->Rows[1]->Cells[0]->Value = Convert::ToString(time\_search\_tree\_chart.count() \* 10000);

dataGridView1->Rows[1]->Cells[1]->Value = found\_quantity\_tree\_chart;

dataGridView1->Rows[1]->Cells[2]->Value = comparison\_tree\_search\_chart / N;

}

}

#pragma endregion

#pragma region Sorting

chart2->Series["Series1"]->Points->Clear(); // Очистка графиков

chart2->Series["Series2"]->Points->Clear();

for (int i = N / 10; i <= N; i = i + N / 10) {

for (int k = 0; k < N; ++k) // Обновление значений массива

arr\_sorting[k] = rand() % 100;

// AVL-сортировка

const auto begin\_tree\_sorting\_chart = chrono::high\_resolution\_clock::now(); // Время начала работы алгоритма

tree = new AVL(i); // Создание нового дерева

int comparisons\_tree\_sort\_chart = 0, permutations\_tree\_chart = 0;

for (int j = 0; j < i; ++j) // Заполнение дерева

tree->add(arr\_sorting[j], comparisons\_tree\_sort\_chart, permutations\_tree\_chart);

const auto end\_tree\_sorting\_chart = chrono::high\_resolution\_clock::now(); // Время завершения работы алгоритма

const auto time\_tree\_sorting\_chart = chrono::duration\_cast<chrono::duration < double>>(end\_tree\_sorting\_chart - begin\_tree\_sorting\_chart);

chart2->Series["Series2"]->Points->AddXY(i, time\_tree\_sorting\_chart.count() \* 10000);

// Сортировка прямым включением

const auto begin\_direct\_sorting\_chart = chrono::high\_resolution\_clock::now();// Время начала работы алгоритма

int comparisons\_direct\_sort\_chart = 0, permutations\_direct\_sort\_chart = 0;

forward\_include(arr\_sorting, i, permutations\_direct\_sort\_chart, comparisons\_direct\_sort\_chart); // Сортировка прямым включением

const auto end\_direct\_sorting\_chart = chrono::high\_resolution\_clock::now(); // Время завершения работы алгоритма

const auto time\_direct\_sorting\_chart = chrono::duration\_cast<chrono::duration < double>>(end\_direct\_sorting\_chart - begin\_direct\_sorting\_chart);

chart2->Series["Series1"]->Points->AddXY(i, time\_direct\_sorting\_chart.count() \* 10000);

if (i == (N / 10) \* 10) {

// Заполнение таблицы с характеристиками поиска

dataGridView2->Rows[0]->Cells[0]->Value = Convert::ToString(time\_direct\_sorting\_chart.count() \* 10000);

dataGridView2->Rows[0]->Cells[1]->Value = comparisons\_direct\_sort\_chart;

dataGridView2->Rows[0]->Cells[2]->Value = permutations\_direct\_sort\_chart;

dataGridView2->Rows[1]->Cells[0]->Value = Convert::ToString(time\_tree\_sorting\_chart.count() \* 10000);

dataGridView2->Rows[1]->Cells[1]->Value = comparisons\_tree\_sort\_chart;

dataGridView2->Rows[1]->Cells[2]->Value = permutations\_tree\_chart;

auto arr\_avl\_sort = new int[N];

tree->left\_to\_right(arr\_avl\_sort, N);

for (int k = 0; k < 100; ++k) {

dataGridView3->Rows[k]->Cells[0]->Value = arr\_sorting[k];

dataGridView3->Rows[k]->Cells[1]->Value = arr\_avl\_sort[k];

}

}

}

#pragma endregion

delete[] arr\_values; // Очистка памяти

delete[] arr\_sorting;

delete[] arr\_keys;

}

// Начать сначала

void MyForm::button6\_click(Object^ sender, EventArgs^ e) {

tree\_demonstration = new AVL();

textBox1->Text = "";

textBox2->Text = "";

textBox3->Text = "[n] - количество повторов \n Вы успешно начали заново";

}

// Оптимальный бинарный поиск

int MyForm::opt\_bin\_looking(int arr[], const int n, const int key, double& comparisons) {

int L = 0, R = n;

while (R > L) {

comparisons += 2;

const int i = (L + R) / 2;

if (arr[i] >= key)

R = i;

else

L = i + 1;

}

comparisons++;

if (arr[R] == key)

return 1;

return 0;

}

// Сортировка прямым включением

void MyForm::forward\_include(int a[], const int n, int& permutations, int& comparisons) {

for (int i = 1; i < n; i++) {

const int data = a[i];

int ind = i;

while (ind > 0 && a[ind - 1] > data) {

a[ind] = a[ind - 1];

ind--;

comparisons++;

}

if (a[ind] != data)

permutations++;

a[ind] = data;

}

}

void MyForm::forward\_include(int a[], const int n) {

for (int i = 1; i < n; i++) {

const int data = a[i];

int ind = i;

while (ind > 0 && a[ind - 1] > data) {

a[ind] = a[ind - 1];

ind--;

}

a[ind] = data;

}

}

**AVL.h**

#pragma once

#include "Node.h"

#include <stack>

#include <string>

using namespace std;

class AVL {

private:

Node\* root = nullptr; // Корень дерева

int count = 0; // Текущее количество узлов

int N; // Размерность дерева

public:

// Конструкторы

AVL();

AVL(int n);

int get\_count() const { return count; }

void set\_N(int N) { this->N = N; }

// Добавление значения

void add(int val);

// Добавление значения (для сортировки)

void add(int val, int& sr, int& per);

// Удаление значения

void remove(int n, string& mess);

// Поиск ключа (с сохранением пути до него)

void avl\_looking(int key, int\* path, bool& is\_found) const;

// Поиск ключа (без сохранения пути до него)

int avl\_search(int key, double& sr) const;

// Обход слева направо. Вернет список наименьших значений

void left\_to\_right(int\* data, int n) const;

// Вывод дерева

void print\_tree(System::Windows::Forms::TextBox^ textBox) const;

private:

// Проверка, нужна ли балансировка

static int need\_balance(Node\* current);

// Восстанавливает корректное значение поля height

static void fix\_height(Node\* current);

// Выполняет балансировку

void balance(Node\* current);

// Левый поворот

void LL (Node\* current);

// Правый поворот

void RR (Node\* current);

// Рекурсивный обход

static void in\_order(const Node\* current, int\* data, const int num, int& c);

};

**AVL.cpp**

#include "AVL.h"

// Конструкторы

AVL::AVL() {

root = nullptr;

count = 1;

N = 10000;

}

AVL::AVL(const int n) {

root = nullptr;

count = 1;

N = n;

}

// Добавление значения

void AVL::add(int val) {

// Дерево пустое

if (root == nullptr)

root = new Node(val);

// Дерево не пустое

else {

Node\* current = root;

bool isEnd = false; // Конец дерева

// Путь к добавленной вершине - false - лево, true - право

bool\* path = new bool[N];

// Переменная, чтобы проверить увеличилась ли высота дерева (была на уровне с новой ячейкой другая)

// Если добавляется уровень - true

bool is\_add = false;

int c = 0;

// Пока не достигнут конец дерева

while (!isEnd) {

// Если значения равны

if (val == current->get\_value()) {

current->set\_same(current->get\_same() + 1);

count++;

isEnd = true;

}

// Если значение меньше корня

else if (val < current->get\_value()) {

// Дошли до конца

if (current->get\_left() == nullptr) {

// Добавили уровень

if (current->get\_right() == nullptr)

is\_add = true;

current->set\_left(new Node(val, current));

count++;

isEnd = true;

}

else {

current = current->get\_left();

path[c] = false;

}

}

// Если значение больше корня

else {

// Дошли до конца

if (current->get\_right() == nullptr) {

// Добавили уровень

if (current->get\_left() == nullptr)

is\_add = true;

current->set\_right(new Node(val, current));

count++;

isEnd = true;

}

else {

current = current->get\_right();

path[c] = true;

}

}

// Если добавили уровень, значит увеличиваем height

if (is\_add) {

current = root;

current->set\_height(current->get\_height() + 1);

// Увеличиваем значение высоты height на всех уровнях

for (int j = 0; j < c; j++) {

if (path[j])

current = current->get\_right();

else

current = current->get\_left();

current->set\_height(current->get\_height() + 1);

}

// Балансировка при необходимости

for (int j = 0; j < c; j++) {

if(abs(need\_balance(current)) >= 2)

balance(current);

current = current->get\_parent();

fix\_height(current);

}

// Для корня

if (abs(need\_balance(current)) >= 2)

balance(current);

fix\_height(current);

}

c++;

}

delete[] path; // Освобождение памяти

}

}

// Добавление значения (для сортировки)

void AVL::add(int val, int& sr, int& per) {

// Дерево пустое

if (root == nullptr)

root = new Node(val);

// Дерево не пустое

else {

Node\* current = root;

bool isEnd = false; // Конец дерева

// Путь к добавленной вершине - false - лево, true - право

bool\* path = new bool[N];

// Переменная, чтобы проверить увеличилась ли высота дерева (была на уровне с новой ячейкой другая)

// Если добавляется уровень - true

bool isAdd = false;

int c = 0;

// Пока не достигнут конец дерева

while (!isEnd) {

// Если значения равны

if (val == current->get\_value()) {

current->set\_same(current->get\_same() + 1);

count++;

isEnd = true;

sr++;

}

// Если значение меньше корня

else if (val < current->get\_value()) {

sr += 2;

// Дошли до конца

if (current->get\_left() == nullptr) {

// Добавили уровень

if (current->get\_right() == nullptr)

isAdd = true;

current->set\_left(new Node(val, current));

count++;

isEnd = true;

}

else {

current = current->get\_left();

path[c] = false;

}

}

// Если значение больше корня

else {

sr += 2;

// Дошли до конца

if (current->get\_right() == nullptr) {

// Добавили уровень

if (current->get\_left() == nullptr)

isAdd = true;

current->set\_right(new Node(val, current));

count++;

isEnd = true;

}

else {

current = current->get\_right();

path[c] = true;

}

}

// Если добавили уровень, значит увеличиваем height

if (isAdd) {

current = root;

current->set\_height(current->get\_height() + 1);

//Увеличиваем значение высоты height на всех уровнях

for (int j = 0; j < c; j++) {

if (path[j])

current = current->get\_right();

else

current = current->get\_left();

current->set\_height(current->get\_height() + 1);

}

// Балансировка при необходимости

for (int j = 0; j < c; j++) {

if (abs(need\_balance(current)) >= 2) {

balance(current);

per++;

}

current = current->get\_parent();

fix\_height(current);

}

// Для корня

if (abs(need\_balance(current)) >= 2) {

balance(current);

per++;

}

fix\_height(current);

}

c++;

}

delete[] path; // освобождение памяти

}

}

// Удаление значения

void AVL::remove(int n, string& mess) {

int\* pa = new int[N]; // 0 - лево, 1 - право

for (int i = 0; i < N; ++i)

pa[i] = -1;

bool is\_found;

avl\_looking(n, pa, is\_found);

if (!is\_found) {

mess = "Значение не найдено! Операция невозможна!";

return;

}

else {

Node\* current = root;

int l = 0;

while (pa[l] != -1) {

if (pa[l] == 1) // Если право

current = current->get\_right();

else if (pa[l] == 0)

current = current->get\_left();

++l;

}

// Есть одинаковые

if (current->get\_same() != 0) {

current->set\_same(current->get\_same() - 1);

return;

}

// Последний элемент

else if (current->get\_left() == nullptr && current->get\_right() == nullptr) {

Node\* temp = current->get\_parent();

if (current == root) {

root = nullptr; // Для корня

return;

}

else if (temp->get\_left() == nullptr) {

current->set\_parent(nullptr);

temp->set\_right(nullptr);

}

else {

current->set\_parent(nullptr);

temp->set\_left(nullptr);

}

for (int i = 0; i < l - 1; i++) {

if (abs(need\_balance(temp)) >= 2)

balance(temp);

temp = temp->get\_parent();

fix\_height(temp);

}

// Для корня

if (abs(need\_balance(temp)) >= 2)

balance(temp);

fix\_height(temp);

}

// Нет левого поддереваа

else if (current->get\_left() == nullptr) {

Node\* temp = current->get\_parent();

if (current == root) {

current->get\_right()->set\_parent(nullptr);

root = current->get\_right();

fix\_height(root);

}

else if (temp->get\_left() == current) {

temp->set\_left(current->get\_right());

current->get\_right()->set\_parent(temp);

current->set\_parent(nullptr);

current->set\_right(nullptr);

}

else {

temp->set\_right(current->get\_right());

current->get\_right()->set\_parent(temp);

current->set\_parent(nullptr);

current->set\_left(nullptr);

}

}

// Нет правого поддерева

else if (current->get\_right() == nullptr) {

Node\* temp = current->get\_parent();

if (current == root) {

current->get\_left()->set\_parent(nullptr);

root = current->get\_left();

fix\_height(root);

}

else if (temp->get\_right() == current) {

temp->set\_right(current->get\_left());

current->get\_left()->set\_parent(temp);

current->set\_parent(nullptr);

current->set\_left(nullptr);

}

else {

temp->set\_left(current->get\_left());

current->get\_left()->set\_parent(temp);

current->set\_parent(nullptr);

current->set\_right(nullptr);

}

}

// Есть 2 поддерева

else {

Node\* temp = current->get\_right();

while (temp->get\_left() != nullptr)

temp = temp->get\_left();

current->set\_value(temp->get\_value());

Node\* temp1 = temp->get\_parent();

if (temp1 == current) {

temp1->get\_right()->set\_parent(nullptr);

temp1->set\_right(nullptr);

}

else {

temp1->set\_left(nullptr);

temp->set\_parent(nullptr);

}

// Балансировка при необходимости

while (temp1->get\_parent() != nullptr) {

if (abs(need\_balance(temp1)) >= 2)

balance(temp1);

temp1 = temp1->get\_parent();

fix\_height(temp1);

}

// Для корня

if (abs(need\_balance(temp1)) >= 2)

balance(temp1);

fix\_height(temp1);

}

}

delete[] pa; // Освобождение памяти

}

// Поиск ключа (с сохранением пути до него)

void AVL::avl\_looking(const int key, int\* path, bool& is\_found) const {

Node\* current = root;

// Есть ли в дереве ключ

bool is\_here = false;

int c = 0;

// Пока ключ не найден

while (!is\_here) {

// Если узла не существует

if (current == nullptr)

break;

// Если ключ найден

else if (current->get\_value() == key)

is\_here = true;

// Если значение в текщем узле меньше ключа

else if (current->get\_value() < key) {

current = current->get\_right();

path[c] = true;

}

// Если значение в текщем узле больше ключа

else if (current->get\_value() > key) {

current = current->get\_left();

path[c] = false;

}

c++;

}

is\_found = is\_here; // Возврат статуса найденности ключа

}

// Поиск ключа (без сохранения пути до него)

int AVL::avl\_search(const int key, double& sr) const {

Node\* current = root;

// Есть ли в дереве ключ

bool is\_here = false;

// Пока ключ не найден

while (!is\_here) {

// Если узла не существует

if (current == nullptr) {

sr++;

break;

}

// Если ключ найден

else if (current->get\_value() == key) {

sr += 2;

is\_here = true;

}

// Если значение в текщем узле меньше ключа

else if (current->get\_value() < key) {

sr += 3;

current = current->get\_right();

}

// Если значение в текщем узле больше ключа

else if (current->get\_value() > key) {

sr += 3;

current = current->get\_left();

}

}

return (is\_here) ? 1 : 0; // Возвращает 1, если ключ найден и 0, если не найден

}

// Обход слева направо. Вернет упорядоченный массив первых n значений

void AVL::left\_to\_right(int\* data, const int n) const {

const Node\* current = root;

for (int i = 0; i < n; ++i)

data[i] = -1;

int count = 0;

in\_order(current, data, n, count);

}

// Вывод дерева

void AVL::print\_tree(System::Windows::Forms::TextBox^ textBox) const {

textBox->Text = "";

stack <Node\*> shared\_stack; // Общий стек для значений дерева

shared\_stack.push(root);

bool empty = false;

string out; // Переменная для вывода

while (!empty) {

stack <Node\*> internal\_stack; // Для задания потомков элемента

empty = true;

while (!shared\_stack.empty()) { // Общий стек не пуст

Node\* temp = shared\_stack.top(); // Переход к следующему элементу

shared\_stack.pop(); // Удаление предыдущего

if (temp != nullptr) {

out.append(to\_string(temp->get\_value())); // Добавление значения элемента в переменную для вывода

out.push\_back('[');

out.append(to\_string(temp->get\_same())); // Добавление количества дубликатов в переменную для вывода

out.push\_back(']');

out.push\_back(';');

internal\_stack.push(temp->get\_left()); // Добавление наследников во внутренний стек

internal\_stack.push(temp->get\_right());

if (temp->get\_left() != nullptr ||

temp->get\_right() != nullptr)

empty = false;

}

else {

out.append("\_\_\_\_; "); // Если значения не найдено

internal\_stack.push(nullptr);

internal\_stack.push(nullptr);

}

}

textBox->Text += gcnew System::String(out.c\_str()) + System::Environment::NewLine; // Вывод одного уровня

out = "";

while (!internal\_stack.empty()) {

shared\_stack.push(internal\_stack.top()); // Перенос элементов из внутреннего стека в общий

internal\_stack.pop();

}

}

}

// Проверка, нужна ли балансировка

inline int AVL::need\_balance(Node\* current) {

if (current == nullptr) return 0;

const int h\_left = (current->get\_left() == nullptr) ? -1 : current->get\_left()->get\_height();

const int h\_right = (current->get\_right() == nullptr) ? -1 : current->get\_right()->get\_height();

return h\_right - h\_left;

}

// Восстанавливает корректное значение поля height

inline void AVL::fix\_height(Node\* current) {

if (current == nullptr) return;

const int h1 = (current->get\_left() == nullptr) ? -1 : current->get\_left()->get\_height();

const int h2 = (current->get\_right() == nullptr) ? -1 : current->get\_right()->get\_height();

current->set\_height((h1 > h2 ? h1 : h2) + 1);

}

// Выполняет балансировку

inline void AVL::balance(Node\* current) {

fix\_height(current);

if (need\_balance(current) == 2) {

if (need\_balance(current->get\_right()) < 0)

RR(current->get\_right());

LL(current);

}

if (need\_balance(current) == -2) {

if (need\_balance(current->get\_left()) > 0)

LL(current->get\_left());

RR(current);

}

}

// Левый поворот

inline void AVL::LL(Node\* current) {

if (current == root) {

current = current->get\_right();

Node\* temp = current->get\_left();

current->set\_left(current->get\_parent());

current->get\_left()->set\_right(nullptr);

current->set\_parent(nullptr);

current->get\_left()->set\_parent(current);

current->get\_left()->set\_right(temp);

if (temp != nullptr)

temp->set\_parent(current->get\_left());

root = current;

}

else {

Node\* cur\_pred = current->get\_parent();

if (cur\_pred->get\_left() != nullptr && cur\_pred->get\_left() == current)

cur\_pred->set\_left(current->get\_right());

else

cur\_pred->set\_right(current->get\_right());

Node\* temp = current->get\_right();

current->set\_parent(temp);

if (temp->get\_left() != nullptr) {

current->set\_right(temp->get\_left());

current->get\_right()->set\_parent(current);

}

else

current->set\_right(nullptr);

temp->set\_parent(cur\_pred);

temp->set\_left(current);

fix\_height(current);

fix\_height(temp);

}

}

// Правый поворот

inline void AVL::RR(Node\* current) {

if (current == root) {

current = current->get\_left();

Node\* temp = current->get\_right();

current->set\_right(current->get\_parent());

current->get\_right()->set\_left(nullptr);

current->set\_parent(nullptr);

current->get\_right()->set\_parent(current);

current->get\_right()->set\_left(temp);

if (temp != nullptr)

temp->set\_parent(current->get\_right());

root = current;

}

else {

Node\* cur\_pred = current->get\_parent();

Node\* temp = current->get\_left();

if (cur\_pred->get\_left() != nullptr && cur\_pred->get\_left() == current)

cur\_pred->set\_left(current->get\_left());

else

cur\_pred->set\_right(current->get\_left());

current->set\_parent(temp);

if (temp->get\_right() != nullptr) {

current->set\_left(temp->get\_right());

current->get\_left()->set\_parent(current);

}

else

current->set\_left(nullptr);

temp->set\_parent(cur\_pred);

temp->set\_right(current);

fix\_height(current);

fix\_height(temp);

}

}

// Рекурсивный обход

void AVL::in\_order(const Node\* current, int\* data, const int num, int& c) {

if (current->get\_left() == nullptr) {

data[c] = current->get\_value();

c++;

for (int i = 0; i < current->get\_same(); i++) {

data[c] = current->get\_value();

c++;

}

if (current->get\_right() != nullptr)

in\_order(current->get\_right(), data, num, c);

return;

}

if (c == num) return;

if (current->get\_left() != nullptr)

in\_order(current->get\_left(), data, num, c);

data[c] = current->get\_value();

c++;

for (int i = 0; i < current->get\_same(); i++) {

data[c] = current->get\_value();

c++;

}

if (c == num) return;

if (current->get\_right() != nullptr)

in\_order(current->get\_right(), data, num, c);

if (c == num) return; }

**Node.h**

#pragma once

class Node {

private:

int value; // Значение

int height; // Высота

Node\* left; // Лево

Node\* right; // Право

Node\* parent; // Родитель

int same; // Количество одинаковых элементов

public:

// Конструктор для 1 значения

Node (const int value);

// Конструктор для всех остальных элементов

Node (const int value, Node\* par);

int get\_value() const { return this->value; }

void set\_value(const int value) { this->value = value; }

int get\_height() const { return this->height; }

void set\_height(const int height) { this->height = height; }

Node\* get\_left() const { return this->left; }

void set\_left(Node\* left) { this->left = left; }

Node\* get\_right() const { return this->right; }

void set\_right(Node\* right) { this->right = right; }

Node\* get\_parent() const { return this->parent; }

void set\_parent(Node\* parent) { this->parent = parent; }

int get\_same() const { return this->same; }

void set\_same(const int same) { this->same = same; }

};

**Node.cpp**

#include "Node.h"

Node::Node(const int value) {

this->value = value;

height = 0;

left = nullptr;

right = nullptr;

parent = nullptr;

same = 0;

}

Node::Node(const int value, Node\* par) {

this->value = value;

height = 0;

left = nullptr;

right = nullptr;

parent = par;

same = 0;

}