

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(ДГТУ)

Факультет «Информатика и вычислительная техника»

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

|  |  |  |
| --- | --- | --- |
|  | | |
| Зав. кафедрой | |  |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | В.В. Долгов |
| (подпись) | |  |
| «\_\_\_» | \_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г. | |

ОТЧЕТ

по учебной ознакомительной практике

на базе ФГБОУ ВО «Донской государственный технический университет»

кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»\_

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н. О. Бояршинов

подпись, дата

Обозначение отчета УП.840000.000 Группа ВПР12

Направление 09.03.04 Программная инженерия

Профиль Программное обеспечение вычислительной техники и автоматизированных систем

Руководитель практики:

от кафедры доцент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.А. Скляренко

подпись, дата

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата подпись

Ростов-на-Дону  
2022



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(ДГТУ)

Факультет «Информатика и вычислительная техника»

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

ЗАДАНИЕ

на выполнение учебной ознакомительной практики

на ФГБОУ ВО «Донской государственный технический университет» \_\_\_

кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

в период с «7» февраля 2022 г. по «11» июня 2022 г.

Обучающийся Бояршинов Никита Олегович

Обозначение отчета УП.840000.000 Группа ВПР12

Срок представления отчета на кафедру «11» июня 2022 г.

Содержание индивидуального задания:

Разработать программное средство для реализации бинарного дерева поиска

|  |  |  |
| --- | --- | --- |
| Руководитель практики от кафедры | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись, дата | доцент, А.А. Скляренко |
| Задание принял к исполнению | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись, дата | Н. О. Бояршинов |



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(ДГТУ)

Факультет «Информатика и вычислительная техника»

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

|  |  |  |
| --- | --- | --- |
| Зав. кафедрой | | «ПОВТ и АС » |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | В.В. Долгов |
| (подпись) | |  |
| «\_\_\_» | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2022 г. | |

Рабочий график (план) проведения практики

|  |  |  |
| --- | --- | --- |
| № | Мероприятие | Срок выполнения |
| 1 | Прохождение вводного и первичного инструктажа по охране труда на рабочем месте, и инструктажа по пожарной безопасности на объекте. | 7.02-11.02 |
| 2 | Аналитический обзор | 12.02-21.03 |
| 3 | Алгоритмическое конструирование | 22.03-10.04 |
| 4 | Программная реализация | 11.04-15.05 |
| 5 | Тестирование приложения | 16.05-25.05 |
| 6 | Подготовка итогового отчета | 26.05-10.06 |
| 7 | Защита итогового отчета | 11.06 |

Руководитель практики:

от кафедры доцент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.А. Скляренко

подпись, дата

Ростов-на-Дону  
2022

ДНЕВНИК ПРОХОЖДЕНИЯ ПРАКТИКИ

|  |  |  |  |
| --- | --- | --- | --- |
| Дата | Место работы | Выполняемые работы | Оценка руководителя |

| 1 | 2 | 3 | 4 |
| --- | --- | --- | --- |
| 7.02-9.02 | кафедра «ПОВТиАС» | Прохождение вводного и первичного инструктажа по охране труда на рабочем месте и по пожарной безопасности на объекте. |  |
| 10.02-11.02 | кафедра «ПОВТиАС» | Разбор организационных требований и методических рекомендаций к оформлению итогового отчета |  |
| 12.02-15.03 | кафедра «ПОВТиАС» | Проведение аналитического обзора и написание первой главы |  |
| 16.03-21.03 | кафедра «ПОВТиАС» | Изучение документации и составление технического задания (приложение А итогового отчета) |  |
| 22.03-10.04 | кафедра «ПОВТиАС» | Алгоритмическое конструирование и написание второй главы |  |
| 11.04-15.05 | кафедра «ПОВТиАС» | Программная реализация. Тестирование и отладка программного средства |  |
| 16.05-20.05 | кафедра «ПОВТиАС» | Написание третей главы «Программное конструирование» |  |
| 21.05-25.05 | кафедра «ПОВТиАС» | Написание четвертой главы «Тестирование программного средства» |  |
| 26.05-10.06 | кафедра «ПОВТиАС» | Подготовка итогового отчета |  |
| 11.06 | кафедра «ПОВТиАС» | Защита итогового отчета |  |

Руководитель практики:

от кафедры доцент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.А. Скляренко

ОТЗЫВ - ХАРАКТЕРИСТИКА

Обучающийся Бояршинов Никита Олегович

1 курса группы ВПР12

кафедры «Программное обеспечение вычислительной техники и автоматизированных систем»

Вид практики: учебная ознакомительная

Наименование места практики: кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

Обучающийся выполнил задания программы практики, разработал графическое приложение для реализации бинарного дерева поиска (методы: поиск, удаление элемента, заполнение дерева и 3 метода обхода дерева), реализовал удобный интуитивно-понятный интерфейс, предусмотрев возможность ввода и вывода веток дерева из файла

Дополнительно ознакомился/изучил различные методы обхода бинарного дерева. Принцип реализации графического интерфейса приложения с помощью библиотеки QT. Также более детально ознакомился с языком программирования C++

Заслуживает оценки \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
|  | Руководитель практики от кафедры  доцент Скляренко А.А.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2022 г. |

Содержание

Изм.

Лист

№ докум.

Подпись

Дата

Лист

6

УП.840000.000

Разраб.

Бояршинов Н.О.

Провер.

Скляренко А.А.

.

Программное средство для реализации бинарного дерева поиска

Лист.

Листов

47

ДГТУ

Кафедра «ПОВТиАС»

[Введение 7](#_Toc105774616)

[1. Бинарное дерево поиска 8](#_Toc105774617)

[1.1 Древовидная структура бинарного дерева 8](#_Toc105774618)

[1.2 Основные методы бинарного дерева поиска 9](#_Toc105774619)

[1.3 Описание операций обхода дерева 10](#_Toc105774620)

[1.4 Постановка задачи 10](#_Toc105774621)

[1.5 Выводы по главе 12](#_Toc105774622)

[2. Алгоритмическое конструирование 13](#_Toc105774623)

[2.1 Общий алгоритм работы программного средства 13](#_Toc105774624)

[2.2 Алгоритм считывания сохраненных узлов 13](#_Toc105774625)

[2.3 Алгоритм сохранения узлов в файл 14](#_Toc105774626)

[2.4 Алгоритм поиска элемента 15](#_Toc105774627)

[2.5 Алгоритм удаления узла 16](#_Toc105774628)

[2.6 Алгоритм добавления узла 17](#_Toc105774629)

[2.7 Алгоритмы обходов бинарного дерева: inorder, preorder, postorder 18](#_Toc105774630)

[2.8 Вывод по главе 18](#_Toc105774630)

[3. Программное конструирование 19](#_Toc105774631)

[3.1 Обоснование выбора средств разработки 19](#_Toc105774632)

[3.2 Описание основных объектов 19](#_Toc105774633)

[3.3 Вывод по главе 21](#_Toc105774634)

[4. Тестирование программного средства 22](#_Toc105774635)

[4.1 Описание контрольных примеров 22](#_Toc105774636)

[4.2 Описание результатов тестирования 23](#_Toc105774637)

[4.3 Вывод по главе 29](#_Toc105774638)

[Заключение 30](#_Toc105774639)

[Перечень использованных информационных ресурсов 31](#_Toc105774640)

[Приложение А Техническое задание 32](#_Toc105774641)

[Приложение Б Исходный код программного средства 37](#_Toc105774642)

Введение

Объектом разработки в учебной ознакомительной практике является структура данных - бинарное дерево поиска.

Целью работы является изучение данной структуры, а затем разработка приложения на языке программирования C++.

Организация данных с помощью бинарных деревьев поиска позволяет значительно сократить время поиска нужного элемента. Поиск элемента в линейных структурах данных обычно осуществляется путем последовательного перебора всех элементов, присутствующих в данной структуре. Поиск по дереву не требует перебора всех элементов, поэтому занимает значительно меньше времени. Бинарное дерево поиска — двоичное дерево, для которого выполняются следующие дополнительные условия: оба поддерева — левое и правое — являются двоичными деревьями поиска; у всех узлов левого поддерева произвольного узла X значения ключей данных меньше, нежели значение ключа данных самого узла X; у всех узлов правого поддерева произвольного узла X значения ключей данных больше либо равны, нежели значение ключа данных самого узла X. данные в каждом узле должны обладать ключами, на которых определена операция сравнения меньше. Как правило, информация, представляющая каждый узел, является записью, а не единственным полем данных.

Основным преимуществом двоичного дерева поиска перед другими структурами данных является возможная высокая эффективность реализации основанных на нём алгоритмов поиска и сортировки. Двоичное дерево поиска применяется для построения более абстрактных структур, таких, как множества, мультимножества, ассоциативные массивы.

1. Бинарное дерево поиска

В данном разделе рассматриваются методы построения и работы бинарного дерева поиска

1.1 Древовидная структура бинарного дерева

Бинарное (двоичное) дерево — это упорядоченное дерево, каждая вершина которого имеет не более двух поддеревьев, причем для каждого узла выполняется правило: в левом поддереве содержатся только ключи, имеющие значения, меньшие, чем значение данного узла, а в правом поддереве содержатся только ключи, имеющие значения, большие, чем значение данного узла.

Бинарное дерево является рекурсивной структурой, поскольку каждое его поддерево само является бинарным деревом и, следовательно, каждый его узел в свою очередь является корнем дерева. Узел дерева, не имеющий потомков, называется листом[1].

Пример бинарного дерева представлен на рисунке 1.1.



Рисунок 1.1 - Схематичное изображение бинарного дерева

Глубина бинарного дерева — это максимальный уровень листа дерева, иначе говоря, длина самого длинного пути от корня к листу дерева.

Бинарное дерево может представлять собой пустое множество или бинарное дерево может выродиться в список (рисунок 1.2), в данном случае такое дерево будет называться вырожденным[3].

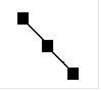


Рисунок 1.2 - Вырожденное бинарное дерево

Бинарное дерево будет представлять собой пустое множество, только в случаях отсутствия корня.

1.2 Основные методы бинарного дерева поиска

Базовый интерфейс двоичного дерева поиска состоит из трёх операций:

* search(k) — поиск узла, значение которого равно k.
* insert(k) — добавление в дерево узла с значением k.
* remove(k) — удаление узла, значение которого равно k.

Этот абстрактный интерфейс является общим случаем, например, таких интерфейсов, взятых из прикладных задач:

* «Телефонная книжка» — хранилище записей (имя человека, его телефон) с операциями поиска и удаления записей по имени человека и операцией добавления новой записи.
* Domain Name Server — хранилище пар (доменное имя, IP адрес) с операциями модификации и поиска.
* Namespace — хранилище имён переменных с их значениями, возникающее в трансляторах языков программирования.

По сути, двоичное дерево поиска — это структура данных, способная хранить таблицу пар (key, value) и поддерживающая три операции: search, insert, remove[2].

1.3 Описание операций обхода дерева

Существуют три операции обхода узлов дерева, отличающиеся порядком обхода узлов. Первая операция — inorder — позволяет обойти все узлы дерева в порядке возрастания ключей. Операция inorder реализована рекурсивным образом: сначала она запускает себя для левого поддерева, потом считывает корень, после чего запускает себя для правого поддерева. Остальные методы обхода дерева реализованы подобным принципом[2].

* inorder — обойти всё дерево, следуя порядку (левое поддерево, вершина, правое поддерево). Элементы по возрастанию
* preorder— обойти всё дерево, следуя порядку (вершина, левое поддерево, правое поддерево). Элементы, как в дереве
* postorder— обойти всё дерево, следуя порядку (левое поддерево, правое поддерево, вершина). Элементы в обратном порядке, как в дереве

1.4 Постановка задачи

На основе проделанного предварительного анализабыли сформулированы следующие принципы работы:

Принцип рационализма. Программный продукт должен иметь все основные инструменты для работы с бинарным деревом, каждый из которых

должен быть реализован в единственном числе.

Принцип устойчивости. Программный продукт создается для использования человеком, поэтому есть вероятность эксплуатационных ошибок. Приложение должно быть устойчиво к любым ошибочным действиям пользователя, для этого необходимо предусмотреть все варианты таких действий и обеспечить корректное реагирование программы на них.

Принцип доступности. Программный продукт предусматривает эксплуатацию пользователем, не имеющим специальных навыков. Приложение должно иметь систему подсказок пользователю и максимально простой, интуитивно-понятный графический интерфейс.

В результате работы была поставлена цель: разработать программное средство, выполняющее функции бинарного дерева поиска.

Был определен набор функциональных требований к разрабатываемому программному продукту, соответствующий нижеперечисленным принципам:

* необходимо реализовать отдельный класс для работы с динамической структурой – бинарное дерево поиска;
* графическое представление дерева;
* поиск узла дерева;
* вставка узла в дерево;
* удаление узла из дерева;
* система ввода и вывода файлов, а также считывания из них узлов дерева;
* inorder обход: вывод отсортированных значений;
* preorder обход: вывод отсортированных значений;
* postorder обход: вывод отсортированных значений;
* корректное реагирование на ошибки пользователей;
* система сообщений об ошибках и подсказок пользователю.

Для создания полноценного программного продукта, в процессе разработки необходимо выполнить все вышеперечисленные функциональные требования. Подробно требования к программному средству представлены в техническом задании (приложение А).

1.5 Выводы по главе

В данной главе была рассмотрена древовидная структура бинарного дерева, рассмотрены понятия: узел, корень, лист. Также были определены основные задачи и методы бинарного дерева поиска. На основе проведенного обзора была поставлена задача и описаны функции, которые будет выполнять разрабатываемое программное средство.

2. Алгоритмическое конструирование

В данном разделе рассматриваются основные алгоритмы работы программного средства: общий алгоритм, алгоритм считывания сохраненных узлов, алгоритм сохранения узлов в файл, основные алгоритмы бинарного дерева: поиск, удаление, вставка узла. Также алгоритмы дополнительных методов бинарного дерева: inorder, preorder, postorder обходы дерева. Приведено их описание и схема работы.

2.1 Общий алгоритм работы программного средства

При запуске программы отображается графический интерфейс с основными блоками: блок логирования, блок графического отображения бинарного дерева, блок пользовательского меню, блок ввода/вывода файлов.

Блок логирования служит для оповещения пользователя о результате операций. Блок графического отображения бинарного дерева служит для отображения текущего состояния элементов дерева. Блок пользовательского меню содержит список доступных пользователю методов. Блок ввода/вывода файлов обеспечивает возможность графически открыть сохраненное дерево или сохранить дерево.

Все вышеперечисленные блоки являются проксирующими методами: связывают методы бинарного дерева поиска с пользовательским интерфейсом.

2.2 Алгоритм считывания сохраненных узлов

Пользователь может выбрать в блоке ввода/вывода файлов пункт открытия файла, выбрать файл с расширением (.bts) или (.txt), после чего выполнится алгоритм считывания сохраненных узлов. Сам алгоритм отображен на рисунке 2.1

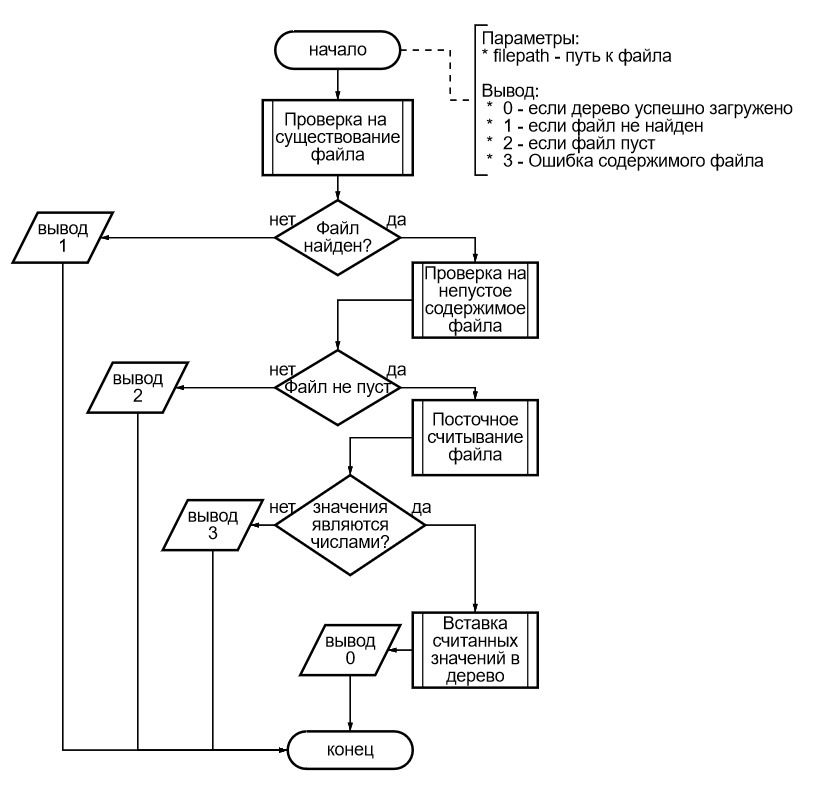


Рисунок 2.1 – Алгоритм считывания сохраненных узлов

2.3 Алгоритм сохранения узлов в файл

Пользователь может выбрать в блоке ввода/вывода файлов пункт сохранения дерева в файл, после чего выбрать путь, куда следует сохранить текущее бинарное древо. Сохранить файл можно только с расширением (.bts). После выбора пути и имени файла, выполняется алгоритм сохранения узлов в файл. Сам алгоритм отображен на рисунке 2.2

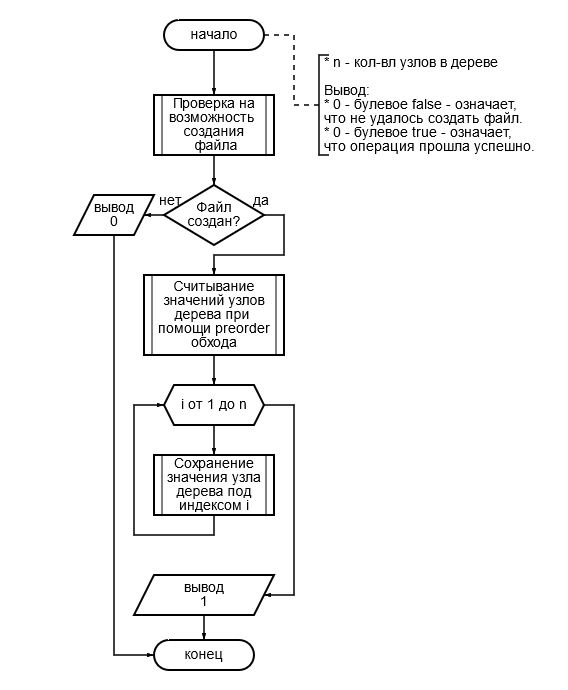


Рисунок 2.2 – Алгоритм сохранения узлов в файл

2.4 Алгоритм поиска элемента

Пользователь может выбрать в блоке пользовательского меню пункт поиска элемента, после чего выполнится алгоритм поиска элемента в бинарном дереве.

При поиске элемента сравнивается искомое значение с корнем. Если искомое больше корня, то поиск продолжается в правом потомке корня, если меньше, то в левом, если равно, то значение найдено и поиск прекращается.

Пошаговый рекурсивный алгоритм поиска элемента в дереве:

Шаг 1. Проверка, является ли искомое значение исходным значением корня. Если да, то возвращаем значение корня и завершаем поиск, если нет, то приступаем к следующему шагу.

Шаг 2. Если искомое значение меньше значения корня, то идем по левому поддереву корня, иначе – по правому, возвращая рекурсивный вызов метода.

Если алгоритм нашел значение, то он возвращает его так, чтобы оно распространялось на каждом шаге рекурсии.

Если алгоритм не нашел значение, значит, мы достигли левого или правого дочернего элемента листового узла, который имеет значение NULL — это значение также рекурсивно распространяется и возвращается.

2.5 Алгоритм удаления узла

Пользователь может выбрать в блоке пользовательского меню пункт удаления узла из бинарного дерева, после чего ввести желаемое значение, которое следует удалить.

Пошаговый рекурсивный алгоритм удаления узла из дерева:

Шаг 1. Если корень пуст, то возвращаем текущую ветку.

Шаг 2. Если искомое значение меньше значения корня, то идем по левому поддереву, а результат присвоим левому поддереву текущего корня.

Шаг 3. Если искомое значение больше значения корня, то идем по правому поддереву, а результат присвоим правому поддереву текущего корня.

Шаг 4. Если же искомое значение является значение текущего корня и левое и правое поддерево текущего корня - не пустое, то находим минимальную ветку в правом поддереве текущего корня, значение которой присваиваем текущему корню, после чего идем по правому поддереву (ищем значение текущего корня), а результат присвоим правому поддереву текущего корня.

Шаг 5. Если же искомое значение является значение текущего корня, но левое или правое поддерево пусто или они пусты одновременно, то:

Если левое поддерево не пустое, а правое пустое, то присваиваем левое поддерево текущего корня текущему корню.

Если левое поддерево пустое, а правое не пустое, то присваиваем правое поддерево текущего корня текущему корню.

Если же и левое и правое поддерево – пусты, то обнуляем текущий корень

Шаг 6. Возвращаем текущий корень.

2.6 Алгоритм добавления узла

Пользователь может выбрать в блоке пользовательского меню пункт добавления узла в дерево, после чего ввести желаемое значение, которое следует добавить.

Пошаговый рекурсивный алгоритм добавления узла в дерево:

Шаг 1. Если текущий корень пуст, то создать новую ветку с значением, которое следует добавить и присвоить созданную ветку текущему корню. После чего вернуть созданную ветку.

Шаг 2. Если значение, которое следует добавить меньше значения текущего корня, то идем по левому поддереву, а результат присвоим левому поддереву текущего корня.

Шаг 3. Если значение, которое следует добавить больше значения текущего корня, то идем по правому поддереву, а результат присвоим правому поддереву текущего корня.

Шаг 4. Если значение, которое следует добавить равно значению текущего корня, то выдать исключение.

Шаг 5. Возвращаем текущий корень.

2.7 Алгоритмы обходов бинарного дерева: inorder, preorder, postorder

Пользователь может выбрать в блоке пользовательского меню пункты вывода отсортированных значений из дерева, после чего выведется отсортированная последовательность элементов дерева. Для выбора доступны

inorder, preorder, postorder обходы. Операция inorder реализована рекурсивным образом: сначала она запускает себя для левого поддерева, потом считывает корень, после чего запускает себя для правого поддерева. Остальные методы обхода дерева реализованы подобным принципом.

1. inorder — обойти всё дерево, следуя порядку: левое поддерево, вершина, правое поддерево. Элементы по возрастанию
2. preorder— обойти всё дерево, следуя порядку: вершина, левое поддерево, правое поддерево. Элементы, как в дереве
3. postorder— обойти всё дерево, следуя порядку: левое поддерево, правое поддерево, вершина. Элементы в обратном порядке, как в дереве

2.8 Выводы по главе

В данной главе были рассмотрены как основные алгоритмы работы программного средства: поиск, удаление, вставка узла; так и дополнительные: inorder, preorder, postorder обходы бинарного дерева и приведены схемы их работы.

Все приведенные алгоритмы используются для дальнейшего проектирования приложения.

3. Программное конструирование

В данном разделе обоснованы выбор языка программирования, используемый для реализации программы, а также представлены основания выбора среды программирования. Определены и описаны основные классы разрабатываемого приложения.

3.1 Обоснование выбора средств разработки

Для разработки программного средства был взят за основу язык программирования C++.

Выбор средства разработки обусловлен учебным планом, удобством в работе с связными списками, благодаря указателям. C++ - компилируемый строго типизированный язык программирования общего назначения. Язык находит свое применение в системном программировании, в частности, при создании операционных систем, драйверов, различных утилит. Данный язык очень хорош для GUI: малое потребление ОЗУ, скорость вычислений.

В качестве среды разработки для базового класса бинарного дерева был взят Clion, для графической составляющей – QtCreator.

3.2 Описание основных объектов

В базовом классе Tree были использованы следующие поля:

* root – указатель на начальный корень дерева

В структуре node, являющейся ячейкой дерева, были использованы следующие поля:

* key – целочисленное значение ячейки (ветки)
* right – указатель на правое поддерево (правая ветка)
* left – указатель на левое поддерево (левая ветка)

Основные реализованные методы базового класса для решения поставленной задачи описываются в таблице 3.1.

Таблица 3.1 – Описание методов базового класса

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Входные параметры | Описание методов | Возвращаемое значение |
| 1 | 2 | 3 | 4 |
| save | Принимает filepath – пусть к файлу, в который следует записать вершины дерева | Записывает в файл preorder отсортированные вершины дерева | Булевое true, если значения успешно записаны, в противном случае false |
| load | Принимает filepath – пусть к файлу, с которого следует считать вершины дерева | Построчно считывает с файла сохраненные вершины и добавляет их в дерево | Число:  0, если дерево успешно загружено;  1, если файл не открылся;  2, если файл пуст;  3, если ошибка содержимого файла; |
| clear | - | Очищает дерево, проходя по каждому элементу дерева | - |
| getRoot | - | Getter; безопасно отдает адрес на вершину дерева(корень), без возможности изменить адрес вершины (корня) | Адрес вершины дерева: root |
| getInorderValues | - | Сортирует значения всех вершин дерева по возрастанию | Вектор из отсортированных вершин дерева (из чисел) |
| getPostorderValues | - | Сортирует значения всех вершин дерева в порядке: левое поддерево, правое поддерево, вершина | Вектор из отсортированных вершин дерева (из чисел) |
| getPreorderValues | - | Сортирует значения всех вершин дерева в порядке: вершина, левое поддерево, правое поддерево | Вектор из отсортированных вершин дерева (из чисел) |

Окончание таблицы 3.1

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| remove | Принимает целочисленное значение, ветку которого следует удалить из дерева | Удаляет ветку с переданным значением из дерева | Булевое true, если значение успешно удалено, в противном случае false |
| search | Принимает целочисленное значение, ветку которого следует найти в дереве | Находит ветку с переданным значением в дереве | Булевое true, если значение успешно найдено, в противном случае false |

Подробный листинг вышеперечисленных подпрограмм приведен в приложении Б.

3.3 Вывод по главе

В данной главе был обоснован выбор языка программирования, а также

среда программирования для создаваемого программного средства. Кроме того, были описаны основные подпрограммы базового класса. В таблице было приведено описание каждой из подпрограмм с указанием типа входных параметров и возвращаемого значения.

4. Тестирование программного средства

В данном разделе описано и протестировано 3 контрольных примера с подробными входными данными и подтверждающими работу программного средства рисунками.

4.1 Описание контрольных примеров

Для тестирования разработанного программного средства было создано несколько контрольных примеров, позволяющих проверить эффективность и корректность работы при различных условиях.

В контрольном примере №1 была поставлена цель протестировать корректность построения бинарного дерево: корректность работы метода добавления новой ветки в пустое, а также в непустое дерево. В качестве входных данных использовались следующие вершины: 345, -321, -325, 400, 2345;

В контрольном примере №2 было построено дерево из вершин: 345; -321; -325; 400; 2345; Структура дерева изображена на рисунке 4.1.

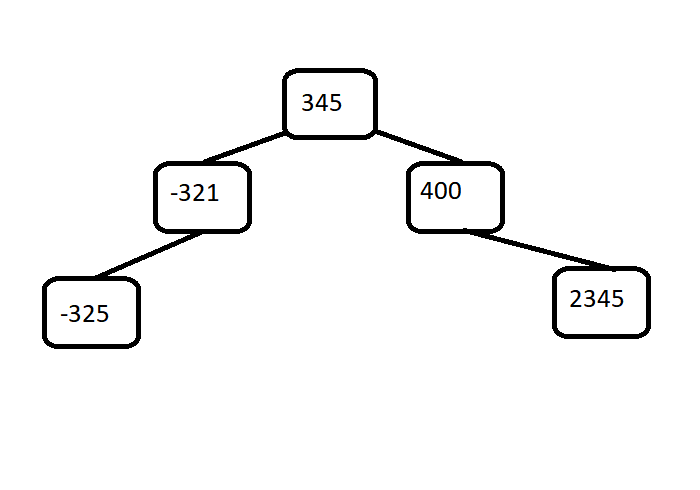


Рисунок 4.1 – Структура дерева с выбранными вершинами

Была поставлена задача протестировать поиск элемента в бинарном дереве. Было рассмотрено 2 случая:

1. Когда вершина существует. В качестве искомого было взято число -321
2. Когда вершины не существует. В качестве искомого было взято число 420

В контрольном примере №3 было построено дерево из вершин: 345; -321; -325; 400; 2345; Структура дерева изображена на рисунке 4.1. Была поставлена задача протестировать корректность удаления ветки из дерева.

Было рассмотрено 2 случая:

1. Когда вершина существует. В качестве удаляемого было взято число 345
2. Когда вершины не существует. В качестве удаляемого взято число 420

В контрольном примере №4 Была поставлена задача протестировать корректность inorder, preorder и postorder обходов дерева.

Было рассмотрено 2 случая:

1. Когда дерево пусто. Дерево не содержит веток
2. Когда дерево содержит ветки. было построено дерево из вершин: 345; -321; -325; 400; 2345; Структура дерева изображена на рисунке 4.1.

4.2 Описание результатов тестирования

Контрольный пример №1: при начальном запуске контрольного примера будет отображено главное меню, где необходимо нажать на кнопку “вставить элемент” после чего откроется окно, куда необходимо ввести значение ветки и нажать кнопку “ok” чтобы вставить новую ветку с введенным значением. Диалоговое окно-сообщение, принимающее числовое значение для вставки новой ветки изображено на рисунке 4.2.

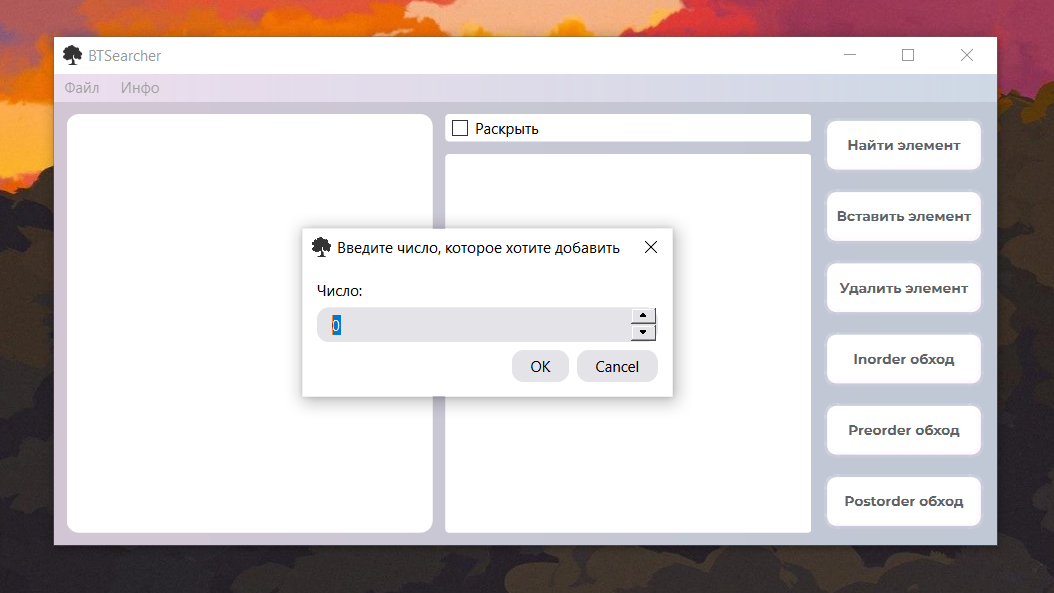


Рисунок 4.2 – Диалоговое окно-сообщение, принимающее числовое значение для вставки новой ветки

После вставки данных (входных) значений в дерево, дерево строится правильно, следуя алгоритму вставки элементов, описанному в главе 2. Состояние программы после вставки входных элементов изображено на рисунке 4.3.

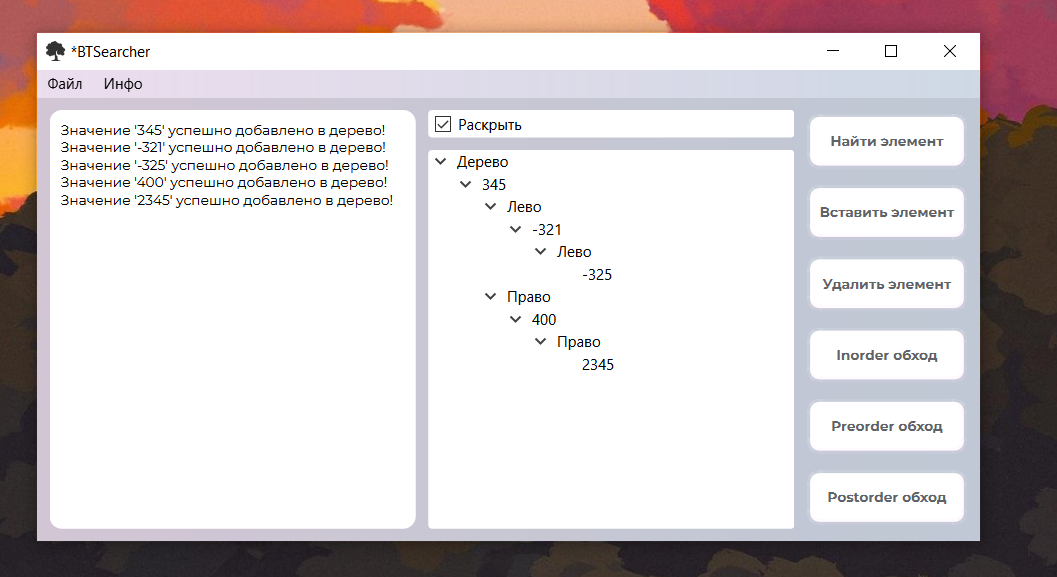


Рисунок 4.3 – Состояние программы после вставки входных элементов

Контрольный пример №2: было построено дерево из вершин: 345; -321; -325; 400; 2345; необходимо нажать на кнопку “Найти элемент”, после чего откроется диалоговое окно, где нужно ввести значение искомого элемента и нажать кнопку “Ок”. Рассматривается 2 случая:

1. Искомая вершина -321. Алгоритм поиска, описанный в главе 2 успешно отработал и нашел элемент, после чего вывел соответствующее сообщение в блок логирования. Результат работы программы изображен на рисунке 4.5.
2. Искомая вершина 420. Алгоритм поиска, описанный в главе 2 успешно отработал и не нашел элемент, значение которого не существует в дереве, выведя в блок логирования соответствующее сообщение об ошибке. Результат работы программы изображен на рисунке 4.6.

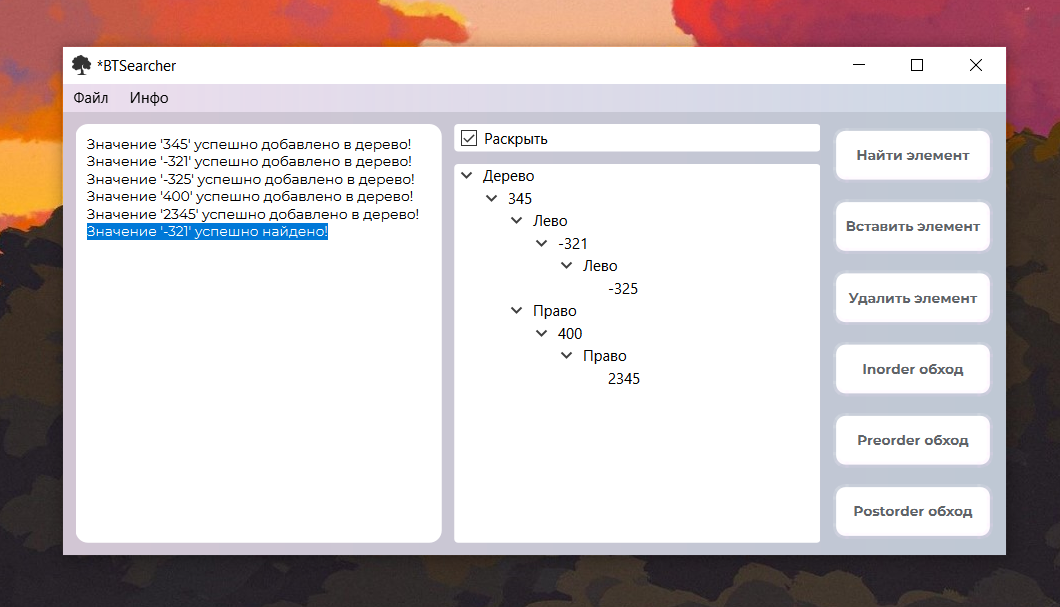


Рисунок 4.5 – Успешно найденный элемент

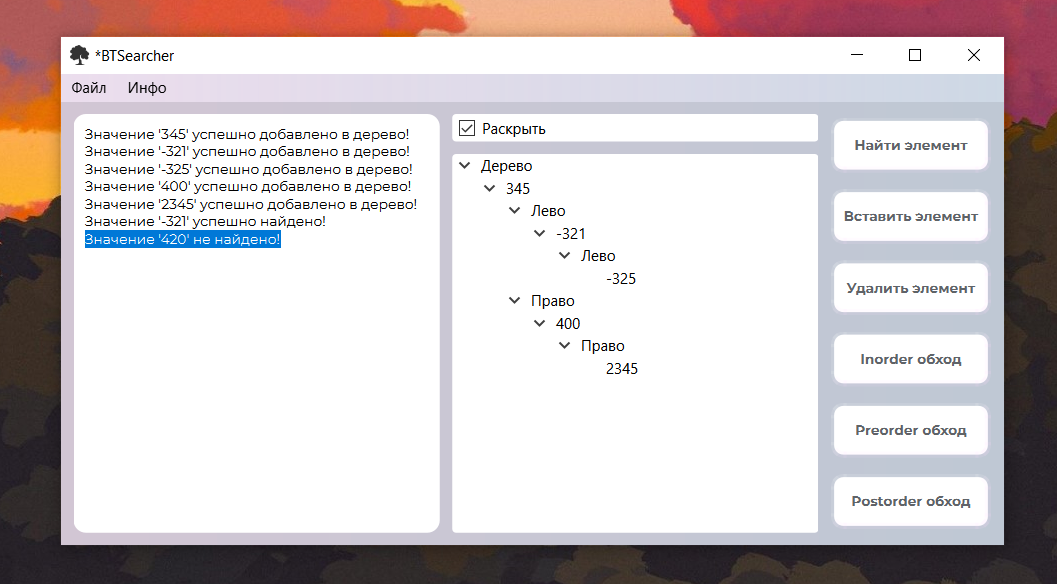


Рисунок 4.6 – Элемент не найден

Контрольный пример №3: было построено дерево из вершин: 345; -321; -325; 400; 2345; необходимо нажать на кнопку “Удалить элемент”, после чего откроется диалоговое окно, где нужно ввести значение удаляемого элемента и нажать кнопку “Ок”. Рассматривается 2 случая:

1. В качестве удаляемого было взято число 345. Алгоритм удаления ветки из дерева, описанный в главе 2 успешно отработал. Было взято минимальное значение в правой ветке, коим оказалась ветка со значением 400, которое и заменило позицию удаляемой ветки, а результат удаления был записан в блок логирования. Результат работы программы изображен на рисунке 4.7.
2. В качестве удаляемого взято число 420. Алгоритм удаления ветки из дерева, описанный в главе 2 успешно отработал. Удаляемое значение не найдено, о чем и сообщила программа в блоке логирования. Результат работы программы изображен на рисунке 4.8.

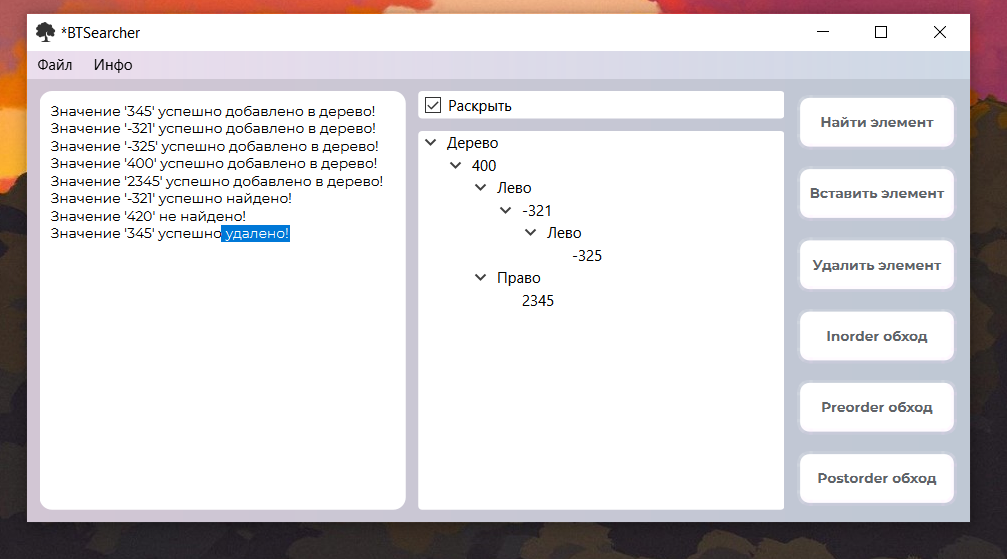


Рисунок 4.7 – Элемент успешно удален

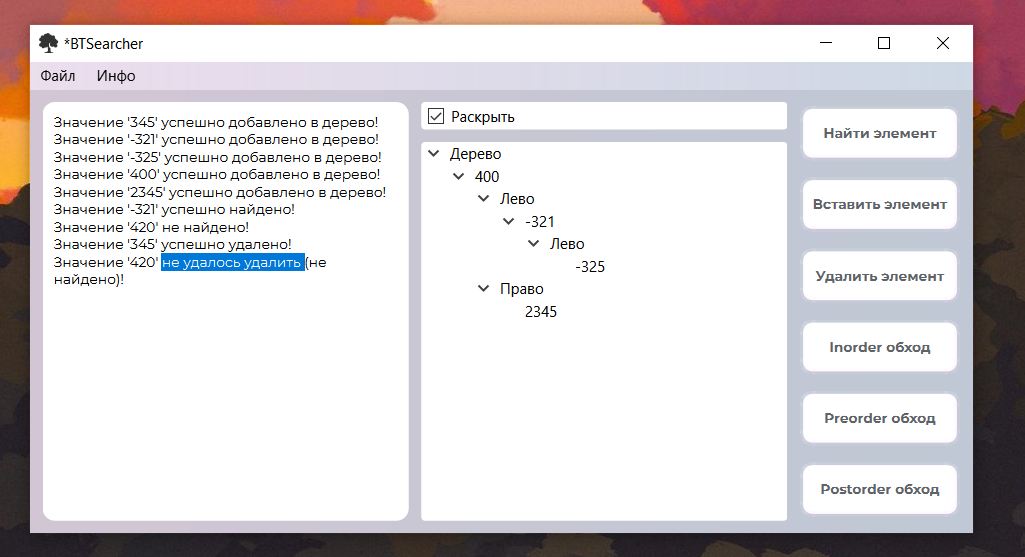


Рисунок 4.8 – Удаляемый элемент не найден

Контрольный пример №4: при начальном запуске контрольного примера будет отображено главное меню, для вывода отсортированных значений следует в блоке меню нажать на кнопки “inorder обход”, “preorder обход” и “postorder обход” – по отдельности, после чего результат обхода будет отражен в блоке логирования. Рассматривается 2 случая:

1. Дерево пусто. Программа вывела соответствующие сообщения в блок логирования. Результат работы программы изображен на рисунке 4.9.
2. Построено дерево из вершин: 345; -321; -325; 400; 2345; Все три алгоритма сортировки, описанные в главе 2 успешно отработали. В блок логирования было выведен ряд сообщений, отражающих успешно выполненную работу программного средства. Результат работы программы изображен на рисунке 4.10.

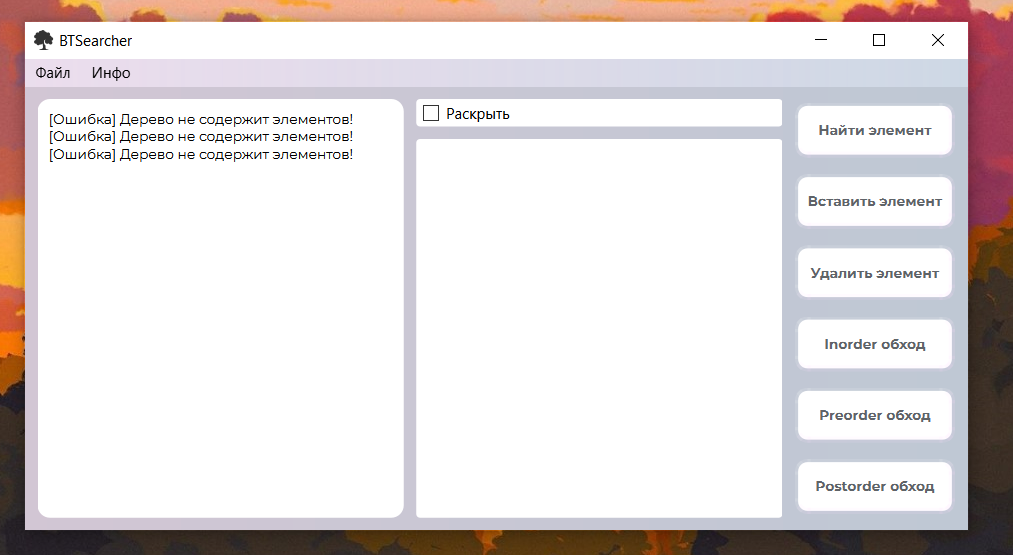


Рисунок 4.9 – Сортируемое дерево пусто

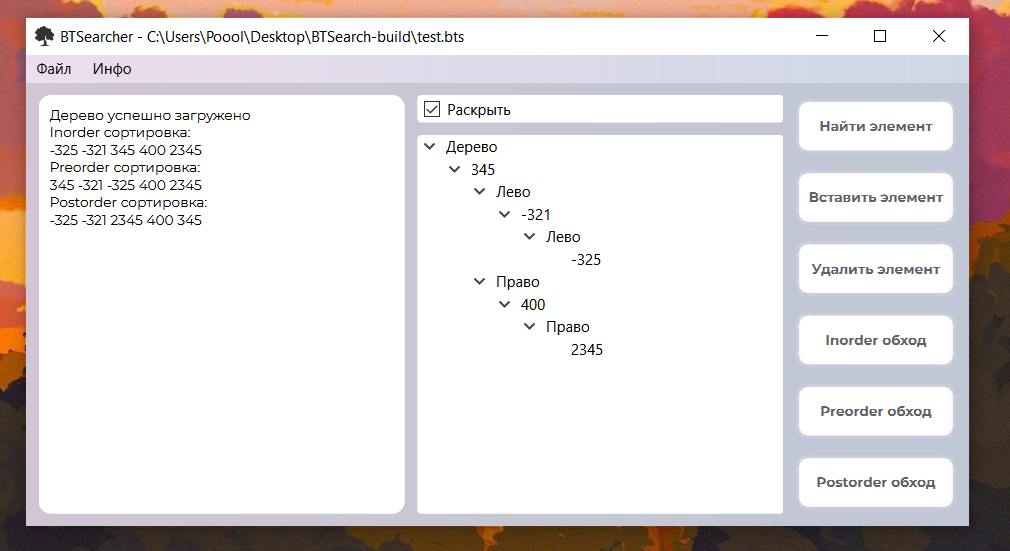


Рисунок 4.10 – Успешно выполненный обход дерева

4.3 Вывод по главе

В данной главе была показана работа программного средства. Программное средство реализует все функции, которые были заданы в техническом задании. Также были решены верно все контрольные примеры. Программное средство работает стабильно.

Заключение

В рамках данной работы был проведен обзор бинарного дерева поиска, были рассмотрены составляющие бинарного дерева, обходы и основные методы.

Был построен аналитически обоснованный алгоритм решения поставленной задачи и было разработано соответствующее программное средство: графическое приложение, эмулирующее бинарное дерево поиска для поиска, удаления, добавления, отображение и сортировки элементов бинарного дерева, конечный исходный код которого можно просмотреть в Приложении Б.

Таким образом была решена поставленная задача и выполнена проверка корректной работоспособности программного средства.

Перечень использованных информационных ресурсов

1. Двоичное дерево поиска (Википедия) [Электронный ресурс] URL:https://ru.wikipedia.org/wiki/Двоичное\_дерево\_поиска
2. Бинарное дерево поиска, справочник ИТМО [Электронный ресурс] URL:<https://neerc.ifmo.ru/wiki/index.php?title=Дерево_поиска,_наивная_реализация>
3. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Алгоритмы: построение и анализ = Introduction to Algorithms / Под ред. И. В. Красикова. — 2-е изд. — М.: Вильямс, 2005. — 1296 с. — ISBN 5-8459-0857-4

Приложение А

Техническое задание

|  |  |
| --- | --- |
| СОГЛАСОВАНО  Доцент каф. «ПОВТиАС»  \_\_\_\_\_\_\_\_\_\_\_/Скляренко А.А  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_2022г. |  |

А.1 Введение

А.1.1 Наименование программного средства

Наименование программного средства – «BTSearcher».

**А.1.2 Область** применения

Образовательная сфера в рамках глубокого изучения тем основанных на алгоритмах бинарного дерева.

А.2 Основание для разработки

Разработка ведется на основании документа «Учебный план для студентов ВУЗа» направление 09.03.04 «Программная инженерия» кафедры «Программное обеспечение вычислительное техники и автоматизированных систем» факультета «Информатика и вычислительная техника» Донского Государственного Технического Университета.

А.3 Назначение разработки

А.3.1 Функциональное назначение

Функциональное назначение программного средства заключается в иллюстрации алгоритмов работы с бинарным деревом.

А.3.2 Эксплуатационное назначение

Эксплуатационное назначение состоит в использовании программного средства на персональном компьютере (ПК) с операционной системой windows.

А.4 Требования к программе

А.4.1 Требования к функциональным характеристикам

Программное средство должно осуществлять следующие функции:

* Вывод и вывод множества вершин из файла;
* Поиск вершины в дереве;
* Удаление элемента из дерева;
* Ручное добавление вершин в дерево;
* Inorder сортировка;
* Preorder сортировка;
* Postorder сортировка.

А.4.2 Требования к надежности

Для надежной работы программного средства необходимо, чтобы выполнялись следующие условия:

* бесперебойное питание технического средства, на котором находится продукт;
* регулярная проверка программного средства на наличие вирусов;
* отсутствие шума и спокойная обстановка.

А.4.2.1 Входные данные

В качестве входных данных являются заданный список вершин дерева.

А.4.2.2 Выходные данные

Выходные данные предоставляются в виде отсортированного preorder списка вершин дерева.

А.4.3 Условия эксплуатации

Для стабильного функционирования и оптимальной работы программного продукта необходимо соблюдение всех требований и правил эксплуатации вычислительной техники. Каких-либо требований к пользователю данного приложения нет.

А.4.4 Требование к составу и параметрам технических средств

В состав технических средств должен входить ПК с операционной системой windows 7 и выше, включающая в себя:

* 64-разрядный процессор;
* оперативная память объемом не менее 1 Гбайт.

Дополнительные требования и ограничения к составу и параметрам технических средств не вводится.

А.4.5 Требование к информационной и программной совместимости

Для функционирования программного средства необходима операционная система Windows 7 и выше.

Пользовательский интерфейс должен быть интуитивно понятным и содержать подсказки.

Исходные коды программы должны быть реализованы на языке c++. В качестве интегрированной среды разработки программы должна быть использована Clion или QtCreator.

А.4.6 Требования к упаковке и маркировки

К упаковке и маркировки специальных требований не предъявляются.

А.4.7 Требования к транспортировке и хранению

Условия транспортирования, места хранения, условия складирования и сроки хранения в различных условиях должны соответствовать требованиям, предъявляемым к носителям информации, на которых будет содержаться данное программное изделие. Программное средство может храниться на любых цифровых носителях информации (жесткий диск, компакт – диск, флэш накопитель и т. п.).

А.5 Требование к программной документации

Программная документация состоит из следующего:

* титульный лист;
* лист задания на учебную ознакомительную практику;
* пояснительная записка к учебной ознакомительной практике;
* техническое задание по ГОСТ 19.201-78 ЕСПД;
* исходный код программного средства по ГОСТ 19.401-79 ЕСПД.

А.6 Стадии и этапы разработки

Реализация программного средства состояла из следующих этапов:

* постановка задачи (7.02-11.02);
* изучение предметной области (12.02-21.03);
* алгоритмическое конструирование (22.03-10.04);
* программная реализация (11.04-15.05);
* тестирование приложения (16.05-25.05);
* разработка отчета (26.05-10.06).

А.7 Порядок и контроль приемки

Порядок и контроль приемки определяются заведующим кафедрой «ПОВТ и АС», подразумевающие собой демонстрацию показателя владения средствами для разработки программных средств в различных направлениях.

Главным требованием к приемке является наличие корректного работающего программного средства и отчета, предоставленного в печатном виде.

|  |  |
| --- | --- |
| Разработчик технического задания:  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2022 г. | /Бояршинов Никита Олегович/  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Приложение Б

Исходный код программного средства

Полный код программного средства находится на съемном носителе, прилагаемом к данной работе.

Листинг Б.1 – Исходный код базового класса

struct node {  
 int key{};  
 struct node \*left{};  
 struct node \*right{};  
};  
  
class Tree {  
 public:  
 node\* getRoot();  
 bool save(const std::string& filepath = "./tree.bts");  
 int load(const std::string& filepath);  
 void clear();  
  
 *// Обходы* std::vector<int> getInorderValues();  
 std::vector<int> getPostorderValues();  
 std::vector<int> getPreorderValues();  
  
 *// Основные методы* bool insert(int value);  
 bool remove(int value);  
 bool search(int value);  
  
  
 private:  
 node\* root;  
  
 bool \_search(node\* \_node, int value);  
 node\* \_add\_node(node\* \_node, int key);  
 node\* \_deleteNode(node\*& \_node, int key);  
  
 *// Обходы дерева* void \_preorder(node \*\_node, std::vector<int> &result);  
 void \_postorder(node \*\_node, std::vector<int> &result);  
 void \_inorder(node \*\_node, std::vector<int> &result);  
  
 *// utils* node\* \_minValueNode(node\* node);  
 bool is\_empty\_file(const std::string& file\_path);  
 bool is\_file\_not\_exist(const std::string& file\_path);  
  
};

node \*Tree::getRoot()  
{  
 return this->root;  
}  
  
bool Tree::save(const std::string& filepath)  
*/\*  
 \* Метод сохраняет  
 \* Дерево в указанный filepath  
 \*/*{  
 std::vector<int> values = getPreorderValues();  
  
 std::ofstream file(filepath);  
 if (file.is\_open())  
 {  
 for (int i = 0; i < values.size(); i++)  
 {  
 file << values[i] << std::endl;  
 }  
 file.close();  
 return true;  
 } else  
 return false;  
}  
  
int Tree::load(const std::string& filepath){  
 if (!this->is\_file\_not\_exist(filepath))  
 {  
 if (!this->is\_empty\_file(filepath))  
 {  
 this->root = nullptr;  
 std::ifstream file(filepath);  
 std::string temp\_str;  
 while (getline(file, temp\_str))  
 {  
 try  
 {  
 (void) std::stoi(temp\_str);  
 }  
 catch (const std::logic\_error &e)  
 {  
 file.close();  
 return 3;  
 }  
 this->insert(std::stoi(temp\_str));  
 }  
 file.close();  
 return 0;  
 } else  
 return 2;  
 } else  
 return 1;  
}  
  
std::vector<int> Tree::getInorderValues(){  
 std::vector<int> result;  
 this->\_inorder(this->root, result);  
 return result;  
}  
  
std::vector<int> Tree::getPreorderValues(){  
 std::vector<int> result;  
 this->\_preorder(this->root, result);  
 return result;  
}  
  
std::vector<int> Tree::getPostorderValues(){  
 std::vector<int> result;  
 this->\_postorder(this->root, result);  
 return result;  
}  
  
  
bool Tree::insert(int value){  
 try  
 {  
 this->root = \_add\_node(this->root, value);  
 }  
 catch( const std::invalid\_argument& error )  
 {  
 return false;  
 }  
 return true;  
}  
  
bool Tree::search(int value){  
 return this->\_search(this->root, value);  
}  
  
bool Tree::remove(int value){  
 if (this->search(value))  
 {  
 this->\_deleteNode(this->root, value);  
 return true;  
 } else  
 return false;  
}  
  
  
void Tree::\_inorder(node \*\_node, std::vector<int> &result){  
 if (\_node != nullptr)  
 {  
 *// Обходим лево* this->\_inorder(\_node->left, result);  
  
 *// Обходим корень* result.push\_back(\_node->key);  
  
 *// Обходим право* this->\_inorder(\_node->right, result);  
  
 }  
}

void Tree::\_preorder(node \*\_node, std::vector<int> &result){  
 if (\_node != nullptr)  
 {  
 result.push\_back(\_node->key);  
this->\_preorder(\_node->left, result);  
this->\_preorder(\_node->right, result);  
 }  
}  
  
void Tree::\_postorder(node \*\_node, std::vector<int> &result){  
 if (\_node != nullptr)  
 {this->\_postorder(\_node->left, result);  
this->\_postorder(\_node->right, result);

result.push\_back(\_node->key);  
 }  
}  
  
  
bool Tree::\_search(node\* \_node, int value)  
{  
 if (\_node == nullptr)  
 return false;  
  
 if (value == \_node->key)  
 return true;  
  
 if (value < \_node->key)  
 return this->\_search(\_node->left, value);  
 if (value > \_node->key)  
 return this->\_search(\_node->right, value);  
 return false;  
}  
  
node \*Tree::\_deleteNode(node \*&\_node, int key)  
{if (\_node == nullptr) return \_node;if (key < \_node->key)  
 \_node->left = this->\_deleteNode(\_node->left, key);  
 else if (key > \_node->key)  
 \_node->right = this->\_deleteNode(\_node->right, key);  
 else if ((\_node->left != nullptr) && (\_node->right != nullptr))  
 {  
 \_node->key = this->\_minValueNode(\_node->right)->key;  
 \_node->right = this->\_deleteNode(\_node->right, \_node->key);  
 } else  
 {  
 if (\_node->left != nullptr)  
 \_node = \_node->left;  
 else if (\_node->right != nullptr)  
 \_node = \_node->right;  
 else  
 \_node = nullptr;  
 }  
 return \_node;  
}

node \*Tree::\_add\_node(node \*\_node, int key)  
{  
 if (\_node == nullptr)  
 {  
 node \*new\_node = new node;  
 new\_node->key = key;  
 return new\_node;  
 }  
  
 if (\_node->key == key)  
 throw std::invalid\_argument("Дерево не может содержать одинаковые значения");

if (key < \_node->key)  
 \_node->left = this->\_add\_node(\_node->left, key);  
 else  
 \_node->right = this->\_add\_node(\_node->right, key);  
  
 return \_node;  
}  
  
void Tree::clear(){  
 if (this->root != nullptr)  
 {  
 std::vector<int> values = this->getPreorderValues();  
 for (int i = 0; i < values.size(); i++)  
 {  
 this->remove(values[i]);  
 }  
 }  
 this->root = nullptr;  
}  
  
  
node \*Tree::\_minValueNode(node \*\_node)  
{  
 struct node \*current = \_node;

while (current && current->left != nullptr)  
 current = current->left;  
  
 return current;  
}  
  
bool Tree::is\_empty\_file(const std::string& file\_path)  
{  
 std::ifstream file(file\_path);  
 file.seekg(0, std::ios::end);  
 long long file\_size = file.tellg();  
 file.close();  
  
 if (file\_size)  
 return false;  
 else  
 return true;  
}

bool Tree::is\_file\_not\_exist(const std::string& file\_path)  
{  
 std::ifstream file(file\_path);  
 bool is\_exist = file.is\_open();  
 file.close();  
  
 if (is\_exist)  
 {  
 return false;  
 }  
 else  
 return true;  
}

Листинг Б.2 – Исходный код графической обработки

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **MainWindow**; }

QT\_END\_NAMESPACE

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

**MainWindow**(QWidget \***parent** = nullptr);

~**MainWindow**();

void **console**(std::string **text**);

private slots:

void **on\_about\_triggered**(bool **checked**);

void **on\_saveAs\_triggered**();

void **on\_openFile\_triggered**();

void **on\_searchInThree\_pressed**();

void **on\_insertInThree\_pressed**();

void **on\_deleteInThree\_pressed**();

void **on\_new\_file\_triggered**();

void **on\_inorder\_pressed**();

void **on\_preorder\_pressed**();

void **on\_postorder\_pressed**();

void **on\_checkBox\_stateChanged**(int **arg1**);

private:

void **setIsEdit**(bool **isEdit**);

void **\_tree\_from\_file\_load**(std::string **filepath**);

void **\_reload\_Qtree**();

void **\_treeViewer**(node \***\_node**, QTreeWidgetItem\* **result** = NULL);

const QString **wTitle** = "BTSearcher";

bool **is\_setting\_three\_uncover** = false;

bool **is\_edit** = false;

Tree **tree** = Tree();

Ui::MainWindow \***ui**;

};

MainWindow::MainWindow(QWidget \*parent): QMainWindow(*parent*), ui(new Ui::MainWindow)

{

ui->setupUi(this);

this->setCentralWidget(*ui*->*horizontalLayoutWidget*);

this->setWindowTitle(this->wTitle);

if (QApplication::arguments().size() > 1)

{

const QString filepath = QApplication::arguments().at(1);

this->\_tree\_from\_file\_load(filepath.toStdString());

}

}

MainWindow::~MainWindow()

{

this->tree.clear();

delete ui;

}

void MainWindow::setIsEdit(bool isEdit)

{

if (this->is\_edit && !isEdit)

{

this->is\_edit = false;

this->setWindowTitle(this->windowTitle().replace("\*", ""));

} else if (!this->is\_edit && isEdit)

{

this->is\_edit = true;

this->setWindowTitle("\*" + this->windowTitle());

}

}

void MainWindow::console(std::string text)

{

this->ui->consoleOutput->append(QString::fromStdString(text));

}

void MainWindow::\_tree\_from\_file\_load(std::string filepath)

{

std::string filename = filepath.substr(filepath.find\_last\_of("/") + 1, std::string::npos);

switch (this->tree.load(filepath))

{

case 0:

this->ui->consoleOutput->clear();

this->\_reload\_Qtree();

this->console("Дерево успешно загружено");

this->setWindowTitle(this->wTitle + " - " + QString::fromStdString(filename));

this->setIsEdit(false);

break;

case 1:

this->console("Ошибка открытия файла! '" + filename + "'");

break;

case 2:

this->console("Файл пуст! '" + filename + "'");

break;

case 3:

this->console("Файл поврежден! '" + filename + "'");

break;

default:

this->console("Неизвестная ошибка! '" + filename + "'");

}

}

void MainWindow::on\_new\_file\_triggered()

{

if (this->is\_edit)

{

QMessageBox msgBox;

msgBox.setText("Хотите сохранить текущее дерево?");

msgBox.setStandardButtons(QMessageBox::Ok | QMessageBox::No | QMessageBox::Cancel);

msgBox.setIcon(QMessageBox::Question);

msgBox.setDefaultButton(QMessageBox::Ok);

int res = msgBox.*exec*();

if (res == QMessageBox::Ok)

this->on\_saveAs\_triggered();

if (res == QMessageBox::Cancel)

return void();

}

this->setIsEdit(false);

this->ui->consoleOutput->clear();

this->tree.clear();

this->\_reload\_Qtree();

this->setWindowTitle(this->wTitle);

}

void MainWindow::on\_about\_triggered(bool checked)

{

QMessageBox aboutBox;

aboutBox.setWindowTitle("О программе");

aboutBox.setText(

"BTSearcher - программа, эмулирующая\n"

"бинарное дерево поиска\n"

"Версия: 1.1.3 \n\n"

"Автор: Бояршинов Н. О.\n"

"Группа: ВПР12\n\n"

"г. Ростов-на-Дону | ДГТУ 2022 год"

);

aboutBox.setIcon(QMessageBox::Information);

aboutBox.setStandardButtons(QMessageBox::Ok);

aboutBox.*exec*();

}

void MainWindow::on\_saveAs\_triggered()

{

QString filepath = QFileDialog::getSaveFileName(this, tr("Сохранить дерево"), "", tr("BTS Файл (\*.bts)"));

if (!filepath.isNull())

{

if (this->tree.save(filepath.toStdString()))

{

this->setIsEdit(false);

this->console("Файл успешно сохранен");

}

else

this->console("Ошибка сохранения файла!");

}

}

void MainWindow::on\_openFile\_triggered()

{

QString filepath = QFileDialog::getOpenFileName(this, tr("Открыть файл дерева"), "", tr("BTS Файлы (\*.bts);; TXT Файлы (\*.txt)"));

if (!filepath.isNull())

{

this->\_tree\_from\_file\_load(filepath.toStdString());

this->setIsEdit(false);

}

}

void MainWindow::on\_searchInThree\_pressed()

{

QInputDialog input\_dialog;

input\_dialog.setWindowTitle("Введите число, которое хотите найти");

input\_dialog.setLabelText("Число:");

input\_dialog.setInputMode(QInputDialog::InputMode::IntInput);

input\_dialog.setIntMinimum(-2147483647);

input\_dialog.setIntMaximum(2147483647);

int result = input\_dialog.*exec*();

if (result == QDialog::Accepted)

{

if (this->tree.search(input\_dialog.intValue()))

this->console("Значение '" + std::to\_string(input\_dialog.intValue()) + "' успешно найдено!");

else

this->console("Значение '" + std::to\_string(input\_dialog.intValue()) + "' не найдено!");

}

}

void MainWindow::on\_insertInThree\_pressed()

{

QInputDialog input\_dialog;

input\_dialog.setWindowTitle("Введите число, которое хотите добавить");

input\_dialog.setLabelText("Число:");

input\_dialog.setInputMode(QInputDialog::InputMode::IntInput);

input\_dialog.setIntMinimum(-2147483647);

input\_dialog.setIntMaximum(2147483647);

int result = input\_dialog.*exec*();

if (result == QDialog::Accepted)

{

if (this->tree.insert(input\_dialog.intValue()))

{

this->console("Значение '" + std::to\_string(input\_dialog.intValue()) + "' успешно добавлено в дерево!");

this->\_reload\_Qtree();

this->setIsEdit(true);

}

else

this->console("Значение '" + std::to\_string(input\_dialog.intValue()) + "' не удалось добавить в дерево (дубликат)!");

}

}

void MainWindow::on\_deleteInThree\_pressed()

{

QInputDialog input\_dialog;

input\_dialog.setWindowTitle("Введите число, которое хотите удалить");

input\_dialog.setLabelText("Число:");

input\_dialog.setInputMode(QInputDialog::InputMode::IntInput);

input\_dialog.setIntMinimum(-2147483647);

input\_dialog.setIntMaximum(2147483647);

int result = input\_dialog.*exec*();

if (result == QDialog::Accepted)

{

if (this->tree.remove(input\_dialog.intValue()))

{

this->console("Значение '" + std::to\_string(input\_dialog.intValue()) + "' успешно удалено!");

this->\_reload\_Qtree();

this->setIsEdit(true);

}

else

this->console("Значение '" + std::to\_string(input\_dialog.intValue()) + "' не удалось удалить (не найдено)!");

}

}

void MainWindow::on\_inorder\_pressed()

{

std::string result;

if (this->tree.getRoot() != NULL)

{

result = "Inorder сортировка:\n";

std::vector<int> values = this->tree.getInorderValues();

for (int i = 0; i < values.size(); i++)

{

result += std::to\_string(values[i]) + " ";

}

} else

result = "[Ошибка] Дерево не содержит элементов!";

this->console(result);

}

void MainWindow::on\_preorder\_pressed()

{

std::string result;

if (this->tree.getRoot() != NULL)

{

result = "Preorder сортировка:\n";

std::vector<int> values = this->tree.getPreorderValues();

for (int i = 0; i < values.size(); i++)

{

result += std::to\_string(values[i]) + " ";

}

} else

result = "[Ошибка] Дерево не содержит элементов!";

this->console(result);

}

void MainWindow::on\_postorder\_pressed()

{

std::string result;

if (this->tree.getRoot() != NULL)

{

result = "Postorder сортировка:\n";

std::vector<int> values = this->tree.getPostorderValues();

for (int i = 0; i < values.size(); i++)

{

result += std::to\_string(values[i]) + " ";

}

} else

result = "[Ошибка] Дерево не содержит элементов!";

this->console(result);

}

void MainWindow::\_reload\_Qtree()

{

ui->treeWidget->clear();

if (this->tree.getRoot() != NULL )

{

QTreeWidgetItem \*item = new QTreeWidgetItem(*ui*->*treeWidget*);

item->setText(0, "Дерево");

this->\_treeViewer(this->tree.getRoot(), *item*);

if (is\_setting\_three\_uncover)

this->ui->treeWidget->expandAll();

}

}

void MainWindow::\_treeViewer(node \*\_node, QTreeWidgetItem\* tree)

{

if (\_node != nullptr)

{

QTreeWidgetItem \*item = new QTreeWidgetItem(*tree*);

item->setText(0, QString::fromStdString(std::to\_string(\_node->key)));

// Обходим лево

QTreeWidgetItem \*item\_left = new QTreeWidgetItem(*item*);

item\_left->setText(0, "Лево");

this->\_treeViewer(*\_node*->*left*, *item\_left*);

if (item\_left->childCount() == 0)

item->removeChild(*item\_left*);

// Обходим право

QTreeWidgetItem \*item\_right = new QTreeWidgetItem(*item*);

item\_right->setText(0, "Право");

this->\_treeViewer(*\_node*->*right*, *item\_right*);

if (item\_right->childCount() == 0)

item->removeChild(*item\_right*);

}

}

void MainWindow::on\_checkBox\_stateChanged(int state)

{

if (state)

this->is\_setting\_three\_uncover = true;

else

this->is\_setting\_three\_uncover = false;

this->\_reload\_Qtree();

}