МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: Демонстрация операций вставки и исключения в идеально сбалансированном бинарном дереве поиска

Студент гр. 9304 _____ Цаплин И.В.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2020

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Цаплин И.В.

Группа 9304

Тема работы: Демонстрация операций вставки и исключения в идеально сбалансированном бинарном дереве поиска

Исходные данные:

Для работы программы требуется мультимедийная бибилиотека SFML

Содержание пояснительной записки:

- Аннотация
- Содержание
- Введение
- Формальная постановка задачи
- Описание алгоритма
- Описание структур данных и функций
- Описание интерфейса пользователя
- Тестирование
- Заключение
- Список использованных источников
- Исходный код программы

Предполагаемый объем пояснительной записки:

Не менее 32 страниц.

Дата выдачи задания: 23.11.2020

Дата сдачи реферата: 23.12.2020

Дата защиты реферата: 24.12.2020

Студент _____ Цаплин И.В.

Преподаватель _____ Филатов А.Ю.

АННОТАЦИЯ

В данной курсовой работе была разработанна программа на языке программирования C++, которая осуществляет визуализацию идеально сбалансированного бинарного дерева поиска. Разработанная в результате выполнения работы программа осуществляет визуализацию полученного идеально сбалансированного бинарного дерева поиска, вставку и удаление заданных элементов и демонстрацию алгоритмов вставки и исключения.

SUMMARY

In this course work, a program is developed in the C++ programming language, which visualizes a perfectly balanced binary search tree. The program developed as a result of the execution of the work visualizes the obtained perfectly balanced binary search tree, inserts and removes specified elements, and demonstrates the algorithms for insertion and exclusion.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Бинарное дерево поиска называется идеально сбалансированным, если для каждой его вершины размеры левого и правого поддеревьев отличаются не более чем на 1.

Цель работы: реализовать идеально сбалансированное бинарное дерево поиска, реализовать операции вставки и исключения элементов. Осуществить демострацию идеально сбалансированного дерева и алгоритмов вставки и исключения.

1. ФОРМАЛЬНАЯ ПОСТАНОВКА ЗАДАЧИ

Реализовать визуализацию идеально сбалансированного бинарного дерева поиска и алгоритмов вставки и исключения элементов. Демонстрация должна быть подробной и понятной (в том числе сопровождаться пояснениями), чтобы программу можно было использовать в обучении для объяснения используемой структуры данных и выполняемых с нею действий.

2. ОПИСАНИЕ АЛГОРИТМА

Программа реализует алгоритм построения идеально сбалансированного бинарного дерева поиска по упорядоченной последовательности данных. Если длина последовательности не равна нулю, создаётся пустой узел дерева. Затем алгоритм выполняется для левой части последовательности, результатом работы является левое поддерево. После создания левого поддерева, текущий элемент последовательности добавляется в созданный пустой узел, алгоритм переходит к следующему элементу последовательности. Затем алгоритм выполняется для правой части последовательности, результатом работы является правое поддерево.

При добавлении и исключении элементов из идеально сбалансированного бинарного требуется перестройка всего дерева. Поэтому алгоритмы добавления и исключения заключаются в добавлении элемента в последовательность или исключении элемента из последовательности и применении алгоритма построения идеально сбалансированного бинарного дерева поиска.

3. ОПИСАНИЕ СТРУКТУР ДАННЫХ И ФУНКЦИЙ

1. Класс BinTreeNode — узел бинарного дерева.

Класс содержит умный указатель на левое поддерево в поле left, умный указатель на правое поддерево в поле right, слабый указатель на родителя в поле parent и данные в поле index. Стуктура класса узла представлена на рисунке 1.

```
class BinTreeNode{
public:
    std::shared_ptr<BinTreeNode> left;
    std::shared_ptr<BinTreeNode> right;
    std::weak_ptr<BinTreeNode> parent;
    int index;
};
```

Рисунок 1 — Структура класса узла бинарного дерева

2. Класс Record — класс для хранения состояний дерева.

Класс содержит вектор бинарных деревьев histroy и вектор строк messages, в которые записываются состояния дерева на каждом шаге выполнения алгоритма и действия, которые выполняются с деревом. Сохранённые состояния и сообщения используются для демонстрации работы алгоритма. Стуктура класса Record представлена на рисунке 2.

```
class Record{
public:
    std::vector<std::string> messages = {};
    std::vector<BinTree> history = {};
};
```

Рисунок 2 — Структура класса записи состояний дерева

3. Класс BinTree - идеально сбалансированное бинарное дерево поиска.

Класс хранит умный указатель на корень дерева в поле head.

Класс содержит упорядоченный вектор элементов дерева в поле elements. Хранить упорядоченную последовательность элементов необходимо, так как она требуется при добавлении и удалении элементов. Для того, чтобы не хранить данные дважды была реализована оптимизация, суть которой

заключается в том, что дерево хранит не сам элемент, а позицию элемента в последовательности elements.

Класс содержит умный указатель на класс Record, в котором хранятся состояния дерева для последующей демонстрации работы алгоритма. Стуктура класса дерева представлена на рисунке 3.

```
class BinTree {
private:
    std::vector<int> elements;
    std::shared_ptr<BinTreeNode> head;
public:
    std::shared_ptr<Record> record;
};
```

Рисунок 3 — Структура класса бинарного дерева

Метод vecToBinTree() - метод, реализующий алгоритм построения идеально сбалансированного бинарного дерева поиска и составляющий историю состояний для демонстрации.

Метод copyBinTree() - метод копирования бинарного дерева.

Метод insertElem() - метод втавки элемента.

Метод deleteElem() - метод удаления элемента.

Метод find() - метод для нахождения элемента в дереве.

Метод printLelel() - метод, печатающий заданный уровень в дереве.

Метод printLelelOrder() - метод, печатающий элементы дерева по уровням.

Метод getHead() - метод, возвращающий указатель на корень дерева.

Метод height() - метод, возвращающий высоту дерева.

Метод drawTree() - метод осуществляет визуализацию дерева.

Метод drawTree() принимает указатель на корень дерева, требуемый размер, координаты и ссылку на окно, в котором должно быть нарисовано дерево. Затем метод рисует корень дерева. Если значение элемента по модулю больше 999, элемент отображается, как «...». Если существует левое или правое

поддеревья, высчитывается необходимый отступ, проводятся линии к координатам поддерева и метод drawTree() вызывается для поддерева.

Метод использует модуль window() для отрисовки окна, в котором демонстируется дерево. Для представления узла используется класс circleShape, который представляет собой окружность. Для отображения элементов и сообщений используется класс Text, который представляет собой текст. Классу Text требуется объект класса Font, который представляет собой шрифт. В программе используется шрифт Arial.

Метод getElements() - возвращает ссылку на упорядоченный вектор элементов.

Метод getElementsString() - возвращает строку, в которой перечислены элементы дерева.

4. Функция checkString() используется для проверки строки для создания дерева на корректность.

4. ОПИСАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

Программа принимает на вход строку из целых чисел, разделенных пробелами. Затем программа принимает элемент, который необходимо добавить в дерево и элемент, который необходимо удалить.

Программа выводит полученное дерево по уровням. Данный вывод используется при тестировании программы.

Затем открывается окно, в котором производится демонстрация работы алгоритма. В начальный момент времени демонстрируется первый шаг алгоритма. Для переключения между шагами используются клавиши «A» и «D». Клавиша «A» - шаг назад, клавиша «D» - шаг вперёд. В демонстрации дерева представлена структура дерева на данном шаге алгоритма, номер шага, сообщение о том, какое действие было совершено на данном шаге, текущий элемент и список элементов, по которому создаётся дерево.

На рисунках 4-13 представлен пример работы программы.

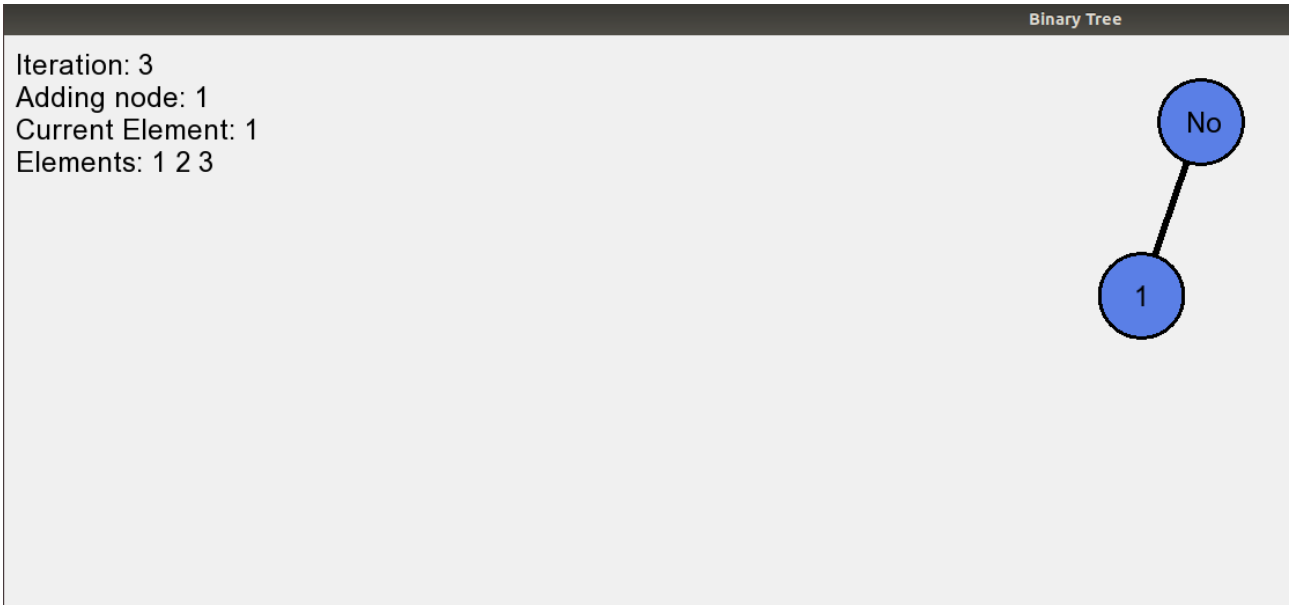

Рисунок 4 — Демонстрация на третьем шаге алгоритма

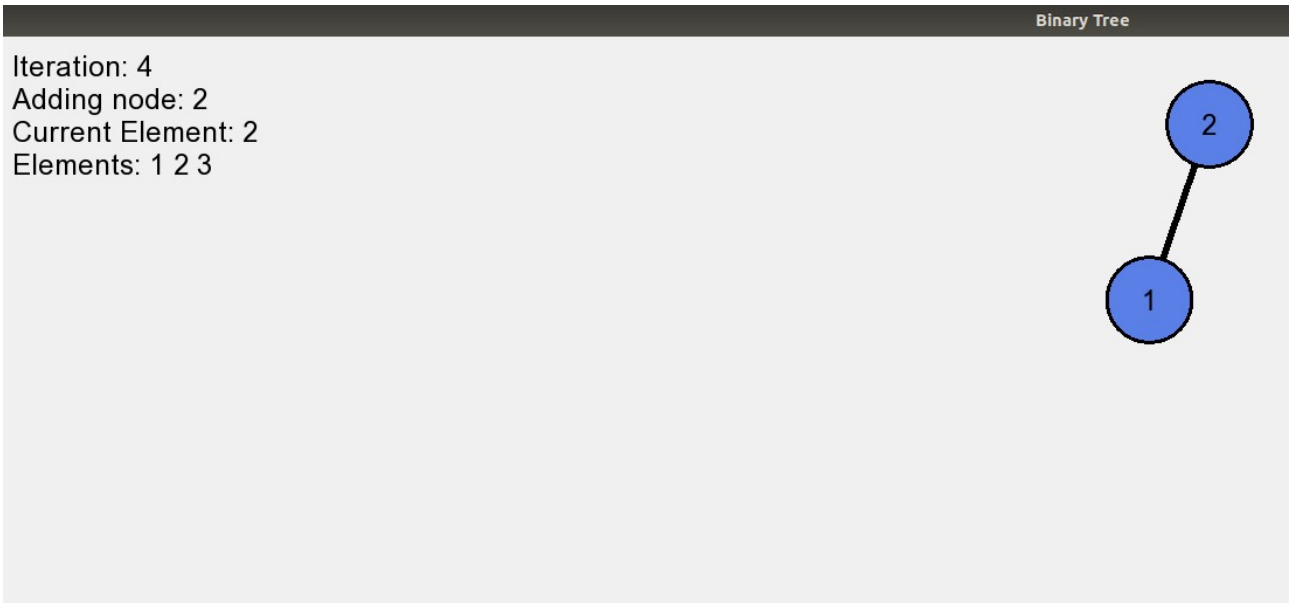

Рисунок 5 — Демонстрация на четвёртом шаге алгоритма

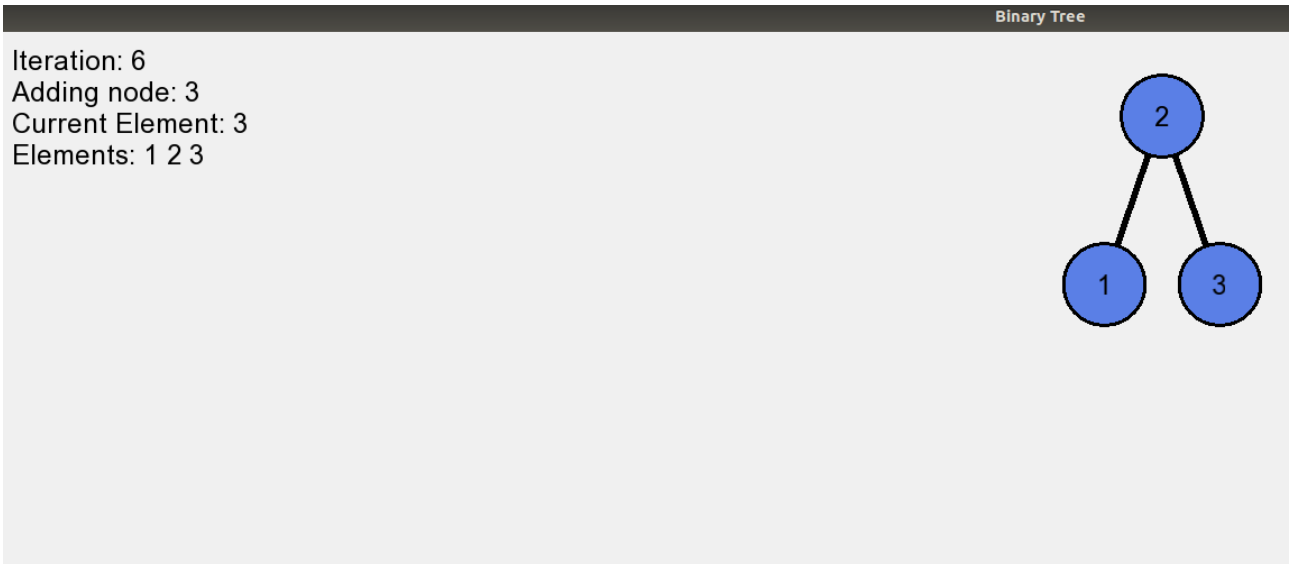

Рисунок 6 — Демонстрация на шестом шаге алгоритма

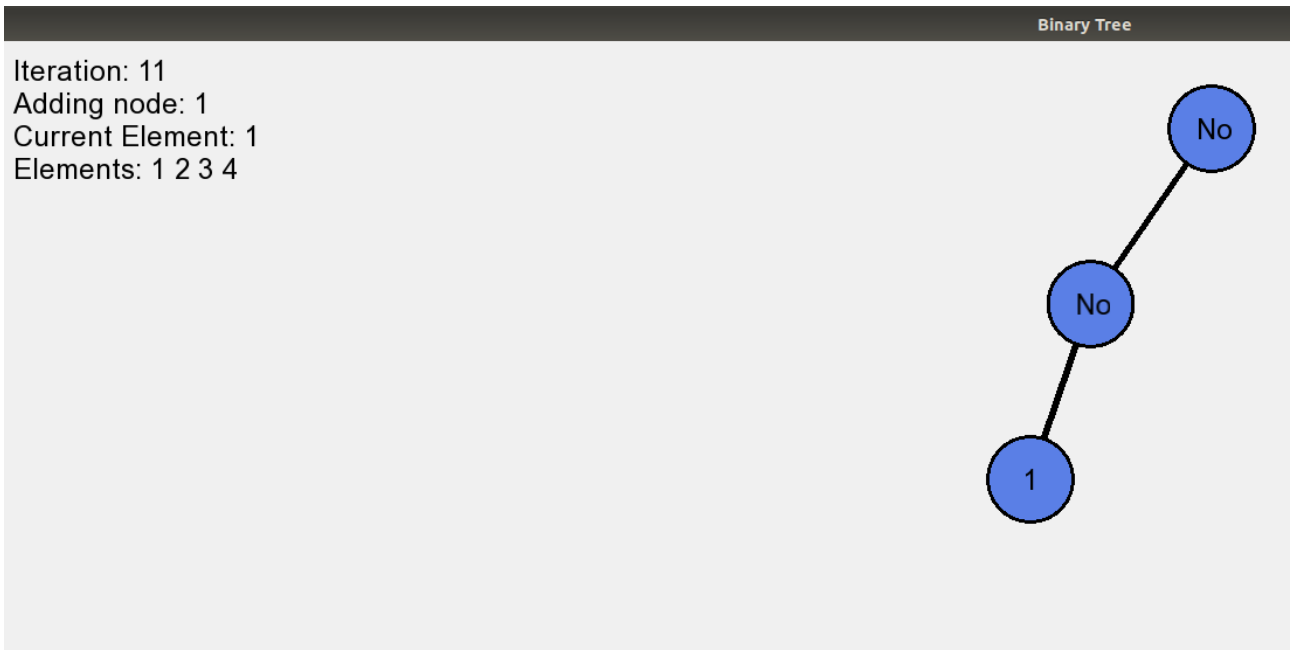

Рисунок 7 — Демонстрация на одиннацатом шаге алгоритма

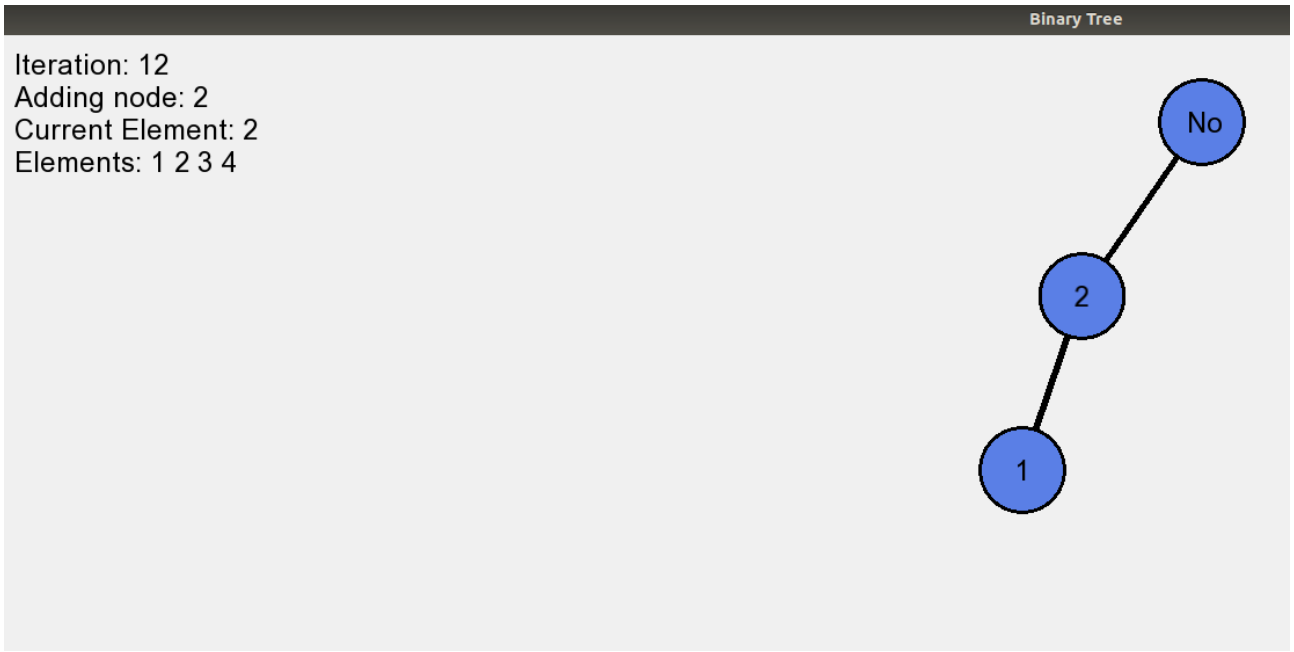

Рисунок 8 — Демонстрация на двенадцатом шаге алгоритма

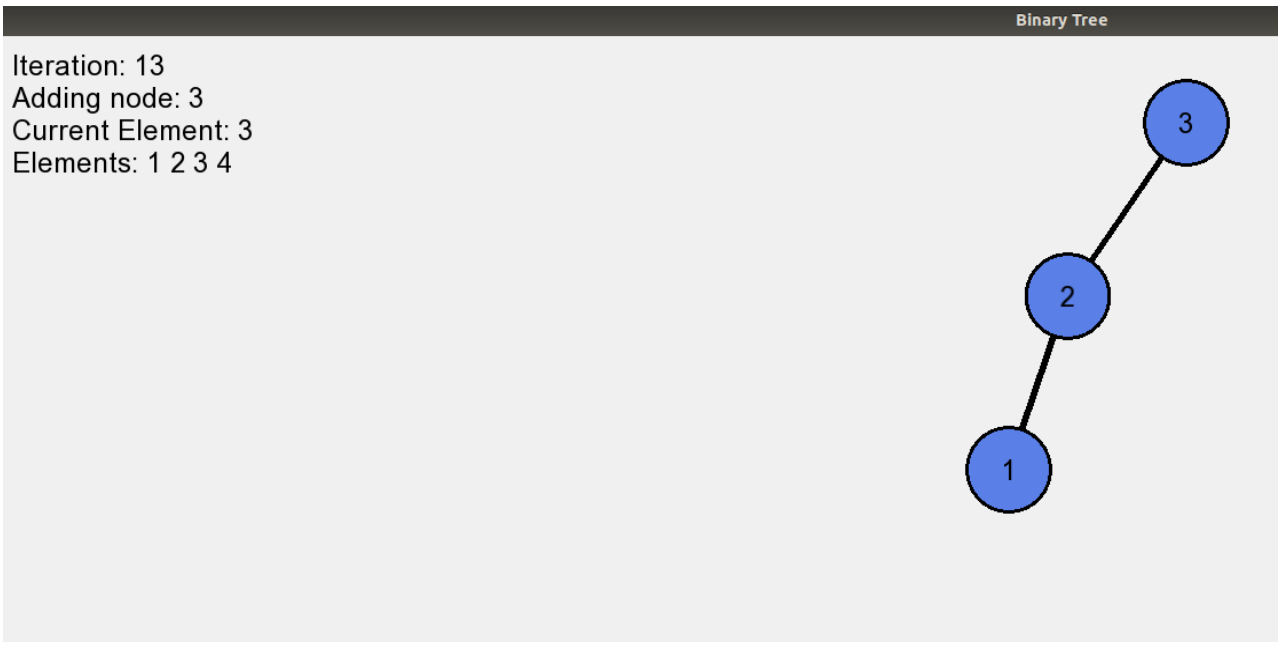

Рисунок 9 — Демонстрация на тринадцатом шаге алгоритма

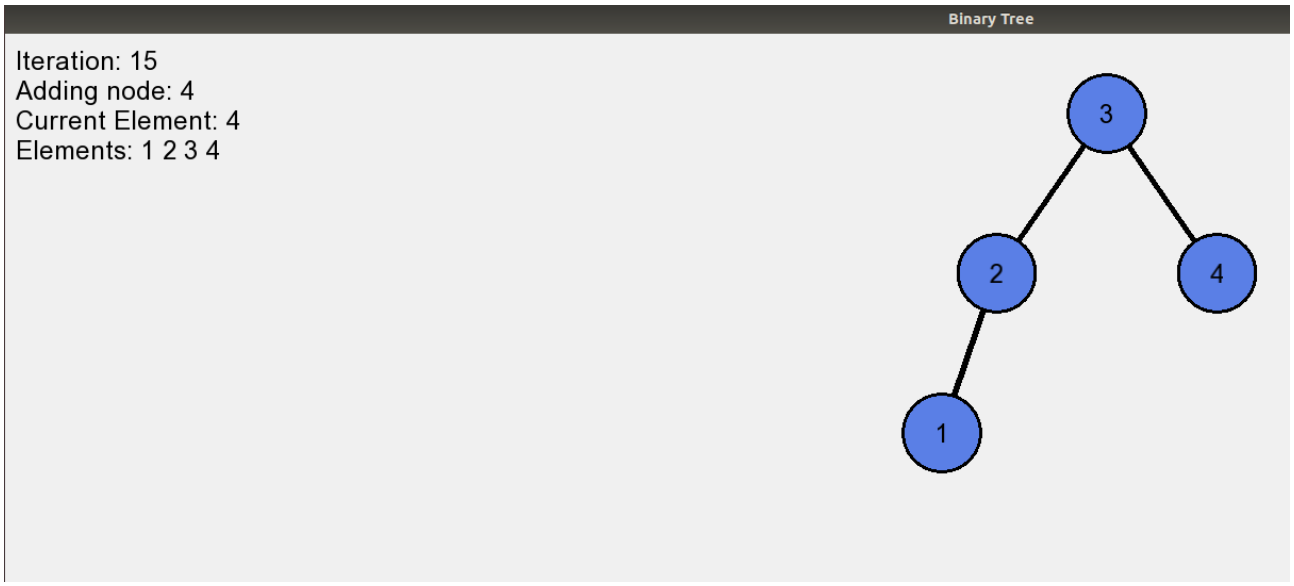

Рисунок 10 — Демонстрация на пятнадцатом шаге алгоритма

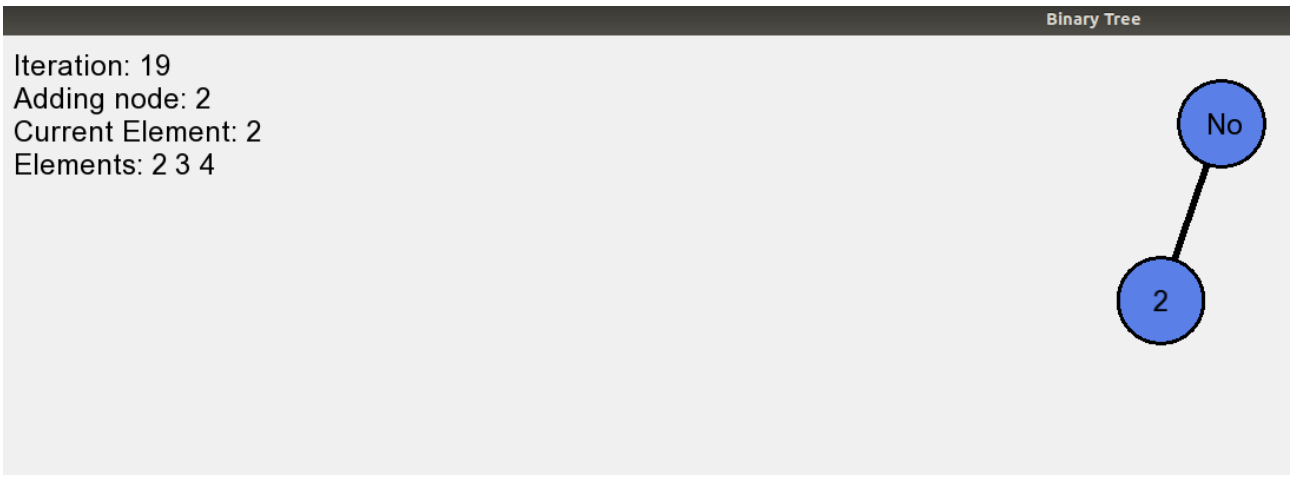

Рисунок 11 — Демонстрация на девятнадцатом шаге алгоритма

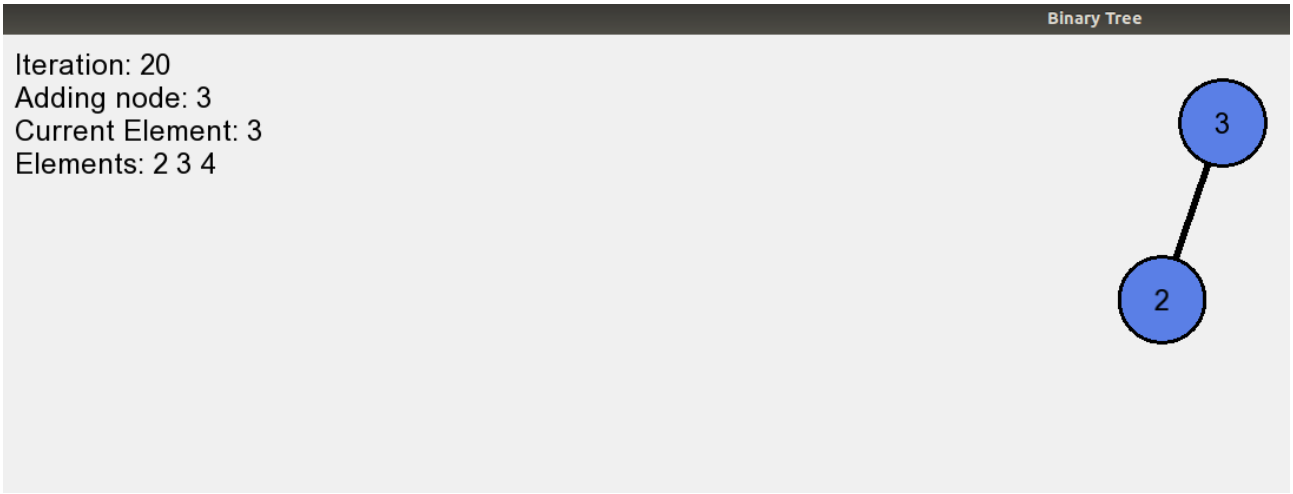

Рисунок 12 — Демонстрация на двадцатом шаге алгоритма

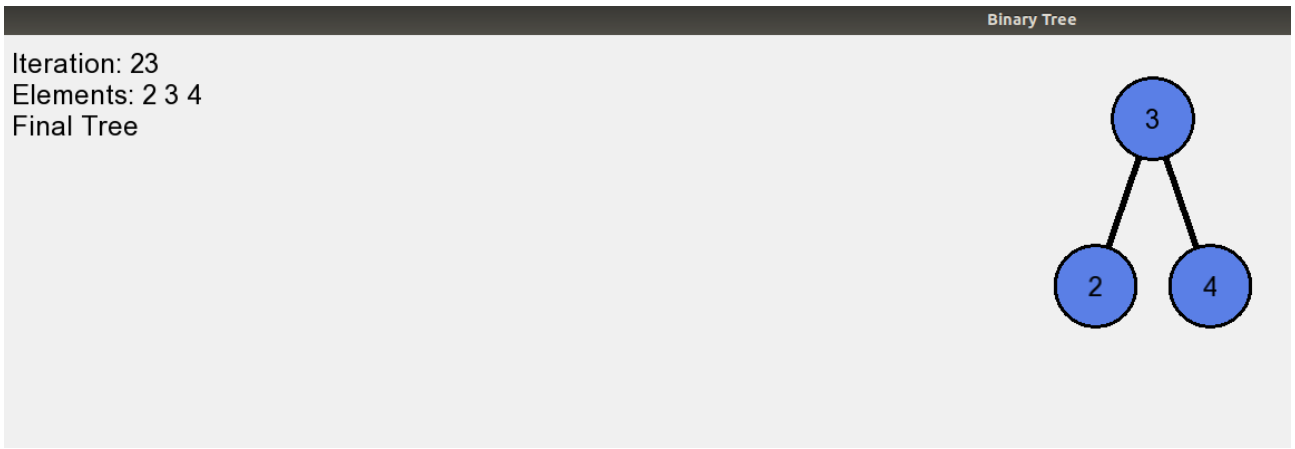

Рисунок 13 — Демонстрация на последнем шаге алгоритма

5. ТЕСТИРОВАНИЕ

Для проведения тестирования был написан bash-скрипт tests_script. Скрипт запускает программу с определёнными входными данными и сравнивает полученные результаты с готовыми ответами. Для каждого теста выводится сообщение TestX <входные данные> passed или TestX <входные данные> failed. Для каждого теста выводится ожидаемый результат и полученный. Полученные в ходе работы файлы с выходными данными удаляются.

Результаты тестирования представлены в таблице 1.

Таблица 1- Результаты тестирования программы

№	Входные данные	Выходные данные
1	1 2 3 4 5 6 3	Enter elements of tree: Enter element to add: Enter element to delete: 4 2 6 1 _ 5 _
2	7 6 5 4 3 2 1 8 4	Enter elements of tree: Enter element to add: Enter element to delete: 5 2 7 1 3 6 8
3	5 5 5 5 5 1 5	Enter elements of tree: Enter element to add: Enter element to delete: 5 5 5 1 _ 5 _
4	1 2 1 2	Enter elements of tree: Enter element to add: Enter element to delete: 1 1 _
5	2 3 4 5 5	Enter elements of tree: Enter element to add: Enter element to delete: 3 2 4
6	10 3 2 8 3 0 6	Enter elements of tree: Enter element

	3 8	to add: Enter element to delete: 3 2 6 0 3 3 10
7	68 30 -4 2 1 -4 -50 3 30 -4	Enter elements of tree: Enter element to add: Enter element to delete: 3 1 30 -4 2 30 68 -50 _ _ _ _ _ _ _

ЗАКЛЮЧЕНИЕ

Идельно сбалансированное бинарное дерево поиска всегда имеет наименьшую высоту возможную для данного числа элементов, поэтому идельно сбалансированное бинарное дерево поиска обеспечивает наименьшее время поиска элемента. При этом при вставке или исключении элемента требуется дорогая операция перестройки всего дерева. По этой причине идеально сбалансированное бинарное дерево поиска целесообразно использовать для данных, для которых часто производится операция поиска элемента, но редко требуется добавлять элемент в дерево или исключать элемент из дерева.

В ходе выполнения работы было реализовано идеально сбалансированное бинарное дерево поиска. Была написана программа на языке программирования C++, демонстрирующая алгоритмы построения дерева, вставки и исключения его элементов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Baeldung. Balanced Binary Search. URL: https:/ /www.baeldung.com (дата обращения: 16.12.2010).

2. Идеально сбалансированные бинарные деревья. URL: http://khpi-iip.mipk.kharkiv.edu/library/datastr/book_sod/kgsu/din_0068.html (дата обращения: 16.12.2010).

3. SFML. URL: https://www.sfml-dev.org/ (дата обращения: 16.12.2010).

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <memory>
#include <iostream>
#include <string>
#include <algorithm>
#include <queue>
#include <sstream>
#include <SFML/Graphics.hpp>
#include <ctgmath>

class BinTree;
class Record{
public:
    std::vector<std::string> messages = {};
    std::vector<BinTree> history = {};
};
class BinTreeNode{
public:
    std::shared_ptr<BinTreeNode> left;
    std::shared_ptr<BinTreeNode> right;
    std::weak_ptr<BinTreeNode> parent;
    int index;
    friend class BinTree;
    BinTreeNode(std::shared_ptr<BinTreeNode> left,
std::shared_ptr<BinTreeNode> right, const
std::shared_ptr<BinTreeNode>& parent , int index):
left(std::move(left)), right(std::move(right)), parent(parent),
index(index)
    {
    }
};

class BinTree {
private:
```

```
        std::vector<int> elements;
        std::shared_ptr<BinTreeNode> head;

        void vecToBinTree(int numOfNodes, int& curIndex,
std::shared_ptr<BinTreeNode> parent) {
            if (numOfNodes == 0) {
                return;
            }
            std::shared_ptr<BinTreeNode> curHead =
std::make_shared<BinTreeNode>(nullptr, nullptr, parent, -1);
            if(numOfNodes == elements.size()){
                head = curHead;
            }
            if(curHead->parent.lock() != nullptr){
                if(curHead->parent.lock()->left == nullptr){
                    curHead->parent.lock()->left = curHead;
                }else{
                    curHead->parent.lock()->right = curHead;
                }
            }
            std::shared_ptr<BinTreeNode> globalHead = curHead;
            while(globalHead->parent.lock() != nullptr){
                globalHead = globalHead->parent.lock();
            }
            BinTree currentState(elements, globalHead);
            std::stringstream stringStream;
            stringStream << "Creating Node " << "\n" << "Current
Element: " << elements[curIndex];
            std::string msg = stringStream.str();
            stringStream.str(std::string());
            record->messages.emplace_back(msg);
            record->history.push_back(currentState);
            vecToBinTree(numOfNodes / 2, curIndex, curHead);
            curHead->index = curIndex;
```

```
            stringStream << "Adding node: " << elements[curIndex]
<< "\n" << "Current Element: " << elements[curIndex];
            BinTree currentState2(elements, globalHead);
            msg = stringStream.str();
            stringStream.str(std::string());
            record->messages.emplace_back(msg);
            record->history.push_back(currentState2);
            curIndex++;
            vecToBinTree(numOfNodes - numOfNodes / 2 - 1,
curIndex, curHead);
        }

        std::shared_ptr<BinTreeNode> copyBinTree(const
std::shared_ptr<BinTreeNode> &otherHead, const
std::shared_ptr<BinTreeNode> &headParent) {
            if (otherHead == nullptr) {
                return nullptr;
            }

            if (otherHead == head) {
                return head;
            }

            std::shared_ptr<BinTreeNode> curHead =
std::make_shared<BinTreeNode>(nullptr, nullptr, headParent,

otherHead->index);
            if (otherHead->left != nullptr) {
                curHead->left = copyBinTree(otherHead->left,
curHead);
            }

            if (otherHead->right != nullptr) {
                curHead->right = copyBinTree(otherHead->right,
curHead);
```

```
            }

            return curHead;
        }

    public:
        std::shared_ptr<Record> record;
        BinTree(std::vector<int> vec) {
            std::sort(vec.begin(), vec.end());
            elements = vec;
            int counter = 0;
            record = std::make_shared<Record>();
            vecToBinTree(elements.size(), counter, nullptr);
        }

        BinTree(std::vector<int>& vec,
std::shared_ptr<BinTreeNode>& otherHead){
            elements = vec;
            record = std::make_shared<Record>();
            this->head = copyBinTree(otherHead, nullptr);
        }

        ~BinTree() = default;

        BinTree(const BinTree &other) {
            head = copyBinTree(other.head, nullptr);
            elements = other.elements;
        }

        BinTree &operator=(const BinTree &other) {
            head = copyBinTree(other.head, nullptr);
            elements = other.elements;
            return *this;
        }
```

```
        BinTree(BinTree &&other) {
            head = std::move(other.head);
            elements = std::move(other.elements);
        }

        BinTree &operator=(BinTree &&other) {
            head = std::move(other.head);
            elements = std::move(other.elements);
            return *this;
        }

        void insertElem(int data) {
            std::stringstream stringStream;
            stringStream << "Adding new element to array: " <<
data << "\nRebuilding tree...";
            std::string msg = stringStream.str();
            BinTree currentState(elements, head);
            elements.push_back(data);
            std::sort(elements.begin(), elements.end());
            record->history.push_back(currentState);
            record->messages.push_back(msg);
            int counter = 0;
            vecToBinTree(elements.size(), counter, nullptr);
        }

        void deleteElem(int data) {
            if(!find(head, data)){
                std::cout << "Element for delete not found" << "\
n";
                return;
            }
            std::stringstream stringStream;
            stringStream << "Deleting element from array: " <<
data << "\nRebuilding tree...";
            std::string msg = stringStream.str();
```

```
            auto position = std::find(elements.begin(),
elements.end(), data);
            BinTree currentState(elements, head);
            elements.erase(position);
            record->history.push_back(currentState);
            record->messages.push_back(msg);
            int counter = 0;
            vecToBinTree(elements.size(), counter, nullptr);
        }

        bool find(const std::shared_ptr<BinTreeNode> &curNode, int
dataToFind) {
            if (curNode == nullptr) {
                return false;
            }
            if (dataToFind == elements[curNode->index]) {
                return true;
            }
            if (dataToFind < elements[curNode->index]) {
                return find(curNode->left, dataToFind);
            }
            return find(curNode->right, dataToFind);
        }

        void printLevelOrder(const std::shared_ptr<BinTreeNode>
&curNode) {
            for (int d = 0; d < this->height(this->getHead()); d+
+) {
                printLevel(curNode, d);
                std::cout << "\n";
            }
        }

        void printLevel(const std::shared_ptr<BinTreeNode>
&curNode, int level) {
```

```
            if (curNode == nullptr) {
                std::cout << "_ ";
                return;
            }
            if (level == 0)
                std::cout << elements[curNode->index] << ' ';
            else {
                if (level > 0) {
                    printLevel(curNode->left, level - 1);
                    printLevel(curNode->right, level - 1);
                }
            }
        }


        std::shared_ptr<BinTreeNode> getHead() {
            return head;
        }

        int height(const std::shared_ptr<BinTreeNode> &curNode) {
            if (curNode == nullptr) {
                return 0;
            }
            if (height(curNode->left) > (height(curNode->right)))
{
                return height(curNode->left) + 1;
            }
            return height(curNode->right) + 1;
        }

        void drawTree(sf::RenderWindow &window, float size, float
x, float y, const std::shared_ptr<BinTreeNode> &curNode) {
            sf::CircleShape node(35.f * size);
            node.setPosition(x, y);
            node.setFillColor(sf::Color(90, 127, 230));
```

```
            node.setOutlineThickness(3);
            node.setOutlineColor(sf::Color::Black);
            sf::Text text;
            sf::Font font;
            font.loadFromFile("./Fonts/arial.ttf");
            text.setFont(font);
            text.setCharacterSize(24 * size);
            if(curNode->index == -1){
                text.setString("No");
                text.setPosition(x + (22 * size), y + (20 *
size));
            }else {
                text.setString(std::to_string(elements[curNode-
>index]));
                if (abs(elements[curNode->index]) < 1000) {
                    if (abs(elements[curNode->index]) < 10) {
                        text.setPosition(x + (28 * size), y + (20
* size));
                    } else {
                        if (abs(elements[curNode->index]) < 100) {
                            text.setPosition(x + (22 * size), y +
(20 * size));
                        } else {
                            text.setPosition(x + (16 * size), y +
(20 * size));
                        }
                    }
                } else {
                    text.setPosition(x + (28 * size), y + (20 *
size));
                    text.setString("...");
                }
            }
            text.setFillColor(sf::Color::Black);
            sf::Text messageBox;
```

```
            if (curNode->left != nullptr) {
                for (int i = -3; i < 3; i++) {
                    sf::Vertex line[] =
                            {
                                    sf::Vertex(sf::Vector2f(x +
(35 * size) + i, y + (35 * size)), sf::Color::Black),
                                    sf::Vertex(sf::Vector2f(
                                            x -
                                            (pow(2, this-
>height(curNode) - 4) * 35 + 17) * size * this->height(curNode) +
                                            (35 * size) + i,
                                            y + (185 * size)),
sf::Color::Black)
                            };
                    window.draw(line, 2, sf::Lines);
                }
                drawTree(window, size, x - (pow(2, this-
>height(curNode) - 4) * 35 + 17) * size * this->height(curNode),
                         y + (150 * size), curNode->left);
            }
            if (curNode->right != nullptr) {
                for (int i = -3; i < 3; i++) {
                    sf::Vertex line[] =
                            {
                                    sf::Vertex(sf::Vector2f(x +
(35 * size) + i, y + (35 * size)), sf::Color::Black),
                                    sf::Vertex(sf::Vector2f(
                                            x +
                                            (pow(2, this-
>height(curNode) - 4) * 35 + 17) * size * this->height(curNode) +
                                            (35 * size) + i,
                                            y + (185 * size)),
sf::Color::Black)
                            };
                    window.draw(line, 2, sf::Lines);
```

```
                }
                drawTree(window, size, x + (pow(2, this-
>height(curNode) - 4) * 35 + 17) * size * this->height(curNode),
                         y + (150 * size), curNode->right);
            }
            window.draw(node);
            window.draw(text);
        }

        std::vector<int>& getElements(){
            return elements;
        }

        std::string getElementsString(){
            std::stringstream stringStream;
            stringStream << "Elements: ";
            for(size_t i = 0; i < getElements().size(); ++i)
            {
                if(i != 0)
                    stringStream << " ";
                stringStream << getElements()[i];
            }
            std::string elementsString = stringStream.str();
            return elementsString;
        }
    };

    bool checkString(std::string& str){
        auto iterator = str.cbegin();
        while(iterator != str.cend()){
            if(*iterator == '-'){
                iterator++;
            }
            if(!isdigit(*iterator)){
                return false;
```

```
            }

            while(isdigit(*iterator)){
                iterator++;
            }

            if((*iterator != ' ') && (iterator != str.cend())){
                return false;
            }

            while(*iterator == ' '){
                iterator++;
            }

        }
        return true;
    }

    int main() {
        std::cout << "Enter elements of tree: ";
        std::string inString{};
        std::getline(std::cin, inString);
        std::vector<int> vecInt;
        if (!checkString(inString)) {
            std::cout << "Incorrect string\n";
            return 0;
        }

        if (!inString.empty()) {
            std::stringstream iss(inString);
            int number;
            while (iss >> number)
                vecInt.push_back(number);
        }
```

```
            std::sort(vecInt.begin(), vecInt.end());
            BinTree tree(vecInt);
            int elemToInsert;
            std::cout << "Enter element to add: ";
            std::cin >> elemToInsert;
            int elemToDelete;
            std::cout << "Enter element to delete: ";
            std::cin >> elemToDelete;
            std::cout << '\n';
            tree.insertElem(elemToInsert);
            tree.deleteElem(elemToDelete);
            tree.printLevelOrder(tree.getHead());
            sf::RenderWindow window;
            window.create(sf::VideoMode(2000, 1000), "Binary Tree");
            int index = 0;
            std::stringstream stringStream;
            std::string msg;
            sf::Font font;
            sf::Text messageBox;
            font.loadFromFile("./Fonts/arial.ttf");
            messageBox.setFont(font);
            messageBox.setCharacterSize(24);
            messageBox.setFillColor(sf::Color::Black);
            messageBox.setPosition(10,10);
            while (window.isOpen())
            {
                sf::Event event{};
                while (window.pollEvent(event))
                {
                    if (event.type == sf::Event::KeyPressed){
                        if
(sf::Keyboard::isKeyPressed(sf::Keyboard::D) && index<tree.record-
>history.size()){
                            index++;
                        }
```

```
                    if
(sf::Keyboard::isKeyPressed(sf::Keyboard::A) && index>0){
                        index--;
                    }
                }
                if (event.type == sf::Event::Closed)
                    window.close();
            }
            window.clear(sf::Color(240,240,240));
            if(index >= tree.record->history.size()) {
                tree.drawTree(window, 1, 1000, 40,
tree.getHead());
                stringStream << "Iteration: " << index+1 << "\n"
<< tree.getElementsString() << "\nFinal Tree\n";
                msg = stringStream.str();
                stringStream.str(std::string());
                messageBox.setString(msg);
                window.draw(messageBox);
            }else{
                tree.record->history[index].drawTree(window, 1,
1000, 40, tree.record->history[index].getHead());
                stringStream << "Iteration: " << index+1 << "\n"
<<tree.record->messages[index] << "\n" << tree.record-
>history[index].getElementsString();
                msg = stringStream.str();
                stringStream.str(std::string());
                messageBox.setString(msg);
                window.draw(messageBox);
            }
            window.display();
        }
    }
```