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

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Алгоритмы и структуры данных»

ТЕМА: ИДЕАЛЬНО СБАЛАНСИРОВАННОЕ БИНАРНОЕ ДЕРЕВО ПОИСКА.

Студентка гр. 9304 _____ Паутова Ю.В.

Преподаватель _____ Филатов Ар.Ю.

Санкт-Петербург
2020

Цель работы.

Ознакомиться с понятием идеально сбалансированного бинарного дерева поиска. Реализовать идеально сбалансированное бинарное дерево поиска на языке программирования C++.

Задание.

Вариант 22

БДП: Идеально сбалансированное; действие:

1. По заданной последовательности элементов Elem построить структуру данных определённого типа – БДП или хеш-таблицу;
2. Для построенной структуры данных проверить, входит ли в неё элемент e типа Elem и если входит, то удалить элемент e из структуры данных (первое обнаруженное вхождение). Предусмотреть возможность повторного выполнения с другим элементом.

Описание алгоритма работы.

Программе с помощью аргументов командной строки подается файл, содержащий 2 строки: последовательность, по которой строится идеально сбалансированное бинарное дерево поиска, и элементы для поиска. Сначала проверяется был ли передан программе файл, если да, то проверяем, открылся ли он для считывания. Если файл был открыт, то обе строки считываются из него и файл закрывается, иначе выводится сообщение об ошибке открытия файла и программа завершается. Если программе не был передан файл для считывания, то строку-дерево предлагается ввести через терминал. Строка-дерево записывается в tree, после чего проверяется с помощью функции check_tree() на то, является она последовательностью чисел (возвращает true) или последовательностью символов/строк (возвращает false). В зависимости от значения, которое вернет функция check_tree(), строка-дерево разбивается по пробелам на элементы и записывается в вектор чисел или вектор строк. Затем есть 2 пути. Первый, если был передан файл, то создается вектор, содержащий элементы для

поиска, и открывается цикл, который завершается, если не были введены элементы для поиска или если были удалены все узлы в дереве, в цикле вызывается метод find() класса BinTree, затем если элемент был найден, вызывается метод deleted() для удаления этого элемента и метод tracking() для вывода дерева. Второй, если не был передан файл, то открывается цикл, который завершается, если в дереве удалены все узлы или было введено «exit», в цикле предлагается ввести элемент для поиска и вызывается метод find() класса BinTree, затем если элемент был найден, вызывается метод deleted() для удаления этого элемента и метод tracking() для вывода дерева.

Формат входных и выходных данных.

Входные данные представлены в виде строки, которая является последовательностью, по которой составляется идеально сбалансированное бинарное дерево поиска. Затем вводится элемент для поиска и удаления.

Выходные данные представлены в виде промежуточных значений при каждом нахождении и удалении введенного элемента.

Описание основных структур данных и функций.

Структуры данных:

- Class BinTreeNode – узел бинарного дерева.
    - Поле std::shared_ptr<BinTreeNode> left – указатель на левое поддерево.
    - Поле std::shared_ptr<BinTreeNode> right – указатель на правое поддерево.
    - Поле Elem data – значение лежащее в узле.
- Class BinTree – реализация идеально сбалансированного бинарного дерева поиска.
    - Поле Elem E – хранит значение элемента, наличие которого нужно проверить и удалить при нахождении.
    - Поле is_find – хранит true, если элемент был найден в дереве.
    - Поле n – хранит количество узлов.

- Поле std::vector<Elem> sequence – хранит последовательность, по которой строится дерево.
- Поле std::shared_ptr<BinTreeNode<Elem>> head – указатель на корень бинарного дерева.
- BinTree(BinTree& other) – конструктор копирования.
- BinTree(BinTree&& other) – конструктор перемещения.
- BinTree& operator=(BinTree& other) – оператор копирования.
- BinTree& operator=(BinTree&& other) – оператор перемещения.
- Метод copy() – копирует бинарное дерево и вовращает указатель на корень.
- Метод make_tree() – создает бинарное дерево.
- Метод make_node() – принимает количество узлов, создает узел.
- Метод printBinTree() – выводит последовательность, по которой построено дерево.
- Метод deleted() – удаляет из дерева узел, значение которого совпадает с введенным значением для поиска.
- Метод tracking() – осуществляет обход дерева в ширину и печатает дерево.
- Метод back_tracking_search() – осуществляет ЛКП-обход и ищет узел, значение которого совпадает с введенным значением для поиска.
- Метод find() – принимает элемент для поиска и вызывает метод back_tracking_search().
- Метод set_sequence() – заполняет поля sequence и n.
- Метод get_head() – возвращает указатель на корень дерева.
- Метод height() – возвращает высоту дерева.
- Метод empty() – проверяет дерево на пустоту.

- Метод back_tracking() – осуществляет ЛКП-обход и заполняет поле sequence.
- Метод deleted_node_left() – удаляет узел из левого поддерева.
- Метод deleted_node_right() – удаляет узел из правого поддерева.

Функции:

- bool check_tree() – принимает ссылку на объект класса string и проверяет является строка последовательностью чисел или символов.
- std::vector<T> read() – принимает ссылку на объект класса string и создает вектор заданного типа T.

Разработанный программный код см. в приложении А.

Тестирование.

Тестирование происходит с помощью bash-скрипта. Он с помощью команд терминала запускает программу, подавая на вход файлы с тестами из директории Tests, и выводит результат. Также запускает программу без указания файла. Для запуска тестирования в консоли используется команда **make run_tests**.

Результаты тестирования см. в приложении В.

Выводы.

Произошло ознакомление с идеально сбалансированным бинарным деревом поиска. Было реализовано идеально сбалансированное бинарное дерево поиска на языке программирования C++.

Была разработана программа, считывающая строку, создающая на её основе вектор и идеально сбалансированное бинарное дерево поиска, определяющая вхождение заданного элемента в дерево и удаляющая его при нахождении. Проведено тестирование программы.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: **source/lab5.cpp**

```
#include <string>
#include <sstream>
#include <fstream>
#include <stack>
#include <vector>
#include <memory>
#include <iostream>
#include <queue>
#include <algorithm>
#include <cmath>

template<typename Elem>
class BinTreeNode{
public:
    Elem data;
    std::shared_ptr<BinTreeNode> left {nullptr};
    std::shared_ptr<BinTreeNode> right {nullptr};
};

template<typename Elem>
class BinTree{
public:
    BinTree() = default;
    ~BinTree() = default;

    BinTree(BinTree&& other){
        std::swap(other.head, this->head);
    }
    BinTree& operator=(BinTree&& other){
        if (&other != this)
            this->head = std::move(other.head);
        return *this;
    }

    BinTree(BinTree& other){
        this->head = copy(other.head);
    }
    BinTree& operator=(BinTree& other){
        if (&other != this)
            this->head = copy(other.head);
        return *this;
    }
    std::shared_ptr<BinTreeNode<Elem>> copy(std::shared_ptr<BinTreeNod
e<Elem>> cur){
        if (cur){
            std::shared_ptr<BinTreeNode<Elem>> node = std::make_shared
<BinTreeNode<Elem>>();
            node->left = copy(cur->left);
            node->right = copy(cur->right);
            node->data = cur->data;
            return node;
        }
```

```
        return nullptr;
    }

    void make_tree(std::vector<Elem> sequence){
        set_sequence(sequence);
        this->head = make_node(this->n);
    }

    void printBinTree(std::shared_ptr<BinTreeNode<Elem>> cur){
        if (cur){
            printBinTree(cur->left);
            std::cout << cur->data << ' ';
            printBinTree(cur->right);
        }
    }

    void deleted(){
        back_tracking(get_head());
        if (this->head->data >= this->E)
            deleted_node_left();
        else
            deleted_node_right();
        this->head = make_node(this->n);
    }

    void tracking(){ //обход в ширину
        if (this->head){
            std::queue<std::shared_ptr<BinTreeNode<Elem>>> Q;
            std::stringstream lower_level;
            std::shared_ptr<BinTreeNode<Elem>> cur = std::make_shared<
BinTreeNode<Elem>>();
            Q.push(this->head);
            int i = 1;
            int j = 0;
            int nodes_at_level = 0;
            int h = get_height();
            std::cout << '\n' << std::string(pow(2, h-i)+1, ' ');
            while(!Q.empty()){
                cur = Q.front();
                if (nodes_at_level == pow(2, i-1)){
                    i += 1;
                    j += 1;
                    std::cout << "\n\n";
                    if (i != h){
                        std::cout << std::string(pow(2, h-i), ' ');
                    }
                    nodes_at_level = 0;
                }
                if (i != h || h == 1){
                    std::cout << cur->data << std::string(pow(2, h-
j)+1, ' ');
                    nodes_at_level += 1;
                }
                else{
                    std::cout << lower_level.str();
                    break;
                }
```

```
                if (cur->left){
                    Q.push(cur->left);
                    if(i == h-1){
                        lower_level << cur->left->data << "  ";
                    }
                }
                else{
                    if(i == h-1){
                        lower_level << "*  ";
                    }
                }
                if(cur->right){
                    Q.push(cur->right);
                    if(i == h-1){
                        lower_level << cur->right->data << "  ";
                    }
                }
                else{
                    if(i == h-1){
                        lower_level << "*  ";
                    }
                }

                Q.pop();
            }
        }
    }

    void back_tracking_search(std::shared_ptr<BinTreeNode<Elem>> cur){
//обход ЛКП
        if (cur){
            if (cur->data == this->E){
                this->is_find = true;
                return;
            }
            if(cur->data > this->E)
                back_tracking_search(cur->left);
            else
                back_tracking_search(cur->right);
        }
    }

    bool find(Elem E){
        this->E = E;
        back_tracking_search(this->head);
        return is_find;
    }
    void set_E(Elem E){
        this->E = E;
    }
    void set_sequence(std::vector<Elem> sequence){
        this->sequence = sequence;
        this->n = sequence.size();
    }
    std::shared_ptr<BinTreeNode<Elem>> get_head(){
        return this->head;
    }
    int get_height(){
```

```
        return ceil(log2(1 + n));
    }
    bool empty(){
        return !n;
    }

private:
    Elem E;
    bool is_find = false;
    int n; // количество узлов
    std::vector<Elem> sequence{};
    std::shared_ptr<BinTreeNode<Elem>> head;

    std::shared_ptr<BinTreeNode<Elem>> make_node(int n){
        if (n == 0){
            return nullptr;
        }
        std::shared_ptr<BinTreeNode<Elem>> cur = std::make_shared<BinT
reeNode<Elem>>();
        cur->left = make_node(n/2);
        cur->data = sequence.front();
        sequence.erase(sequence.begin());
        cur->right = make_node(n - (n/2) - 1);
        return cur;
    }

    void back_tracking(std::shared_ptr<BinTreeNode<Elem>> cur){ //обхо
д ЛКП
        if (cur){
            back_tracking(cur->left);
            this->sequence.push_back(cur->data);
            back_tracking(cur->right);
        }
    }

    void deleted_node_left(){
        for (int i = (n/2); i >= 0; i--){
            if (this->E == sequence[i]){
                sequence.erase(sequence.begin()+i);
                n -= 1;
                break;
            }
        }
    }
    void deleted_node_right(){
        for (int i = (n/2+1); i < n; i++){
            if (this->E == sequence[i]){
                sequence.erase(sequence.begin()+i);
                n -= 1;
                break;
            }
        }
    }
};

template<typename T>
std::vector<T> read(std::string& tree){
    std::stringstream ss(tree);
```

```
    std::vector<T> array{};
    T value;

    while(ss >> value){
        array.push_back(value);
        if (ss.peek() == ' '){
            ss.ignore();
        }
        if(ss.peek() == '\n'){
            break;
        }
    }
    Std::sort(array.begin(), array.end());
    return array;
}

bool check_tree(std::string& argument);

int main(int argc, char** argv){
    std::string tree;
    std::string to_find;
    bool is_file = false;
    if (argc < 2){
        std::cout << "tree = ";
        std::getline(std::cin, tree);
    }
    else{
        std::ifstream in(argv[1]);
        if (in.is_open()){
            is_file = true;
            std::getline(in, tree);
            std::getline(in, to_find);
            std::cout << "tree = " << tree << "\n";
        }
        else{
            std::cout << "Faild to open " << argv[1] << std::endl;
            return 1;
        }
        in.close();
    }
    bool all_is_number = check_tree(tree);
    bool end = false;
    if (all_is_number){
        BinTree<int> bin_tree;
        bin_tree.make_tree(read<int>(tree));
        bin_tree.tracking();
        if (is_file){
            std::vector<int> data_to_find = read<int>(to_find);
            while (!end){
                if (data_to_find.empty() || bin_tree.empty()){
                    end = true;
                }
                else{
                    if (bin_tree.find(data_to_find.front())){
                        std::cout << "\nElement to find and to delete:
 " << data_to_find.front();
                        bin_tree.deleted();
                        bin_tree.tracking();
```

```
                    }
                    data_to_find.erase(data_to_find.begin());
                }
            }
        }
        else{
            while(!end && !bin_tree.empty()){
                std::cout << "\nInput element to find or exit\nInput:
";
                std::getline(std::cin, to_find);
                if (to_find == "exit"){
                    end = true;
                }
                else{
                    if(bin_tree.find(std::stoi(to_find))){
                        bin_tree.deleted();
                        bin_tree.tracking();
                    }
                }
            }
        }
    }
    else{
        BinTree<std::string> bin_tree;
        bin_tree.make_tree(read<std::string>(tree));
        bin_tree.tracking();
        if (is_file){
            std::vector<std::string> data_to_find = read<std::string>(
to_find);
            while (!end){
                if (data_to_find.empty() || bin_tree.empty()){
                    end = true;
                }
                else{
                    if (bin_tree.find(data_to_find.front())){
                        std::cout << "\nElement to find and to delete:
 " << data_to_find.front();
                        bin_tree.deleted();
                        bin_tree.tracking();
                    }
                    data_to_find.erase(data_to_find.begin());
                }
            }
        }
        else{
            while(!end && !bin_tree.empty()){
                std::cout << "\nInput element to find or exit\nInput:
";
                std::getline(std::cin, to_find);
                if (to_find == "exit"){
                    end = true;
                }
                else{
                    if(bin_tree.find(to_find)){
                        bin_tree.deleted();
                        bin_tree.tracking();
                    }
                }
```

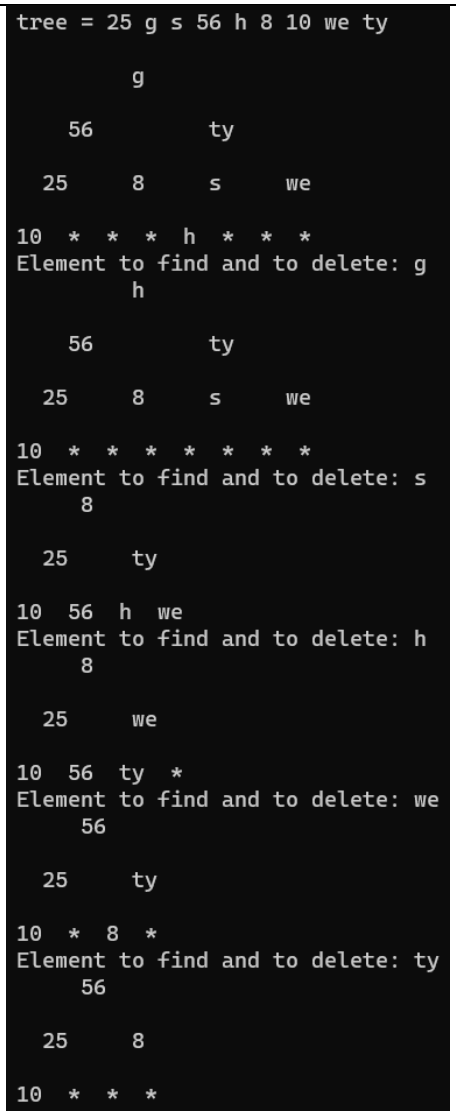
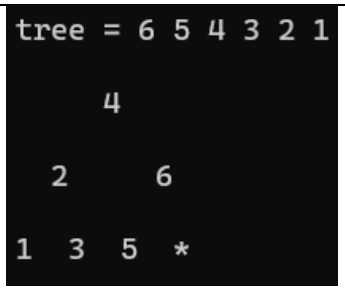
```
            }
        }
    }
    std::cout << std::endl;
    return 0;
}

bool check_tree(std::string& argument){
    auto iterator = argument.cbegin();
    while(iterator != argument.cend()){
        if(*iterator == '-'){
            iterator++;
        }
        if(!isdigit(*iterator)){
            return false;
        }
        while(isdigit(*iterator)){
            iterator++;
        }
        if((*iterator != ' ') && (iterator != argument.cend()) && (*it
erator != '.')){
            return false;
        }
        while(*iterator == ' '){
            iterator++;
        }
    }
    return true;
}
```

ПРИЛОЖЕНИЕ В
ТЕСТИРОВАНИЕ

Таблица В.1 – Результаты тестирования

№	Входные данные	Выходные данные	Комментарии
1	1 2 3 4 5 6 7 8 9 1 9 5	tree = 1 2 3 4 5 6 7 8 9 5 3 8 2 4 7 9 1 * * * 6 * * * Element to find and to delete: 1 6 4 8 3 5 7 9 2 * * * * * * * Element to find and to delete: 9 5 3 7 2 4 6 8 Element to find and to delete: 5 6 3 8 2 4 7 *	Удалены из дерева узлы со значениями: 1, 9 и 5.
2	f t y s e n o y e o	tree = f t y s e n o o f t e n s y Element to find and to delete: y o f t e n s * Element to find and to delete: e o n t f * s * Element to find and to delete: o s n t f * * *	Удалены из дерева узлы со значениями: y, e и o.

3	-1 2 -3 14 -5 6 -7 8 -9 10 -1 -3 -5 -7 -9		tree = -1 2 -3 14 -5 6 -7 8 -9 10 2 -5 10 -7 -1 8 14 -9 * -3 * 6 * * * Element to find and to delete: -1 2 -5 10 -7 -3 8 14 -9 * * * 6 * * * Element to find and to delete: -3 6 -5 10 -7 2 8 14 -9 * * * * * * * Element to find and to delete: -5 6 -7 10 -9 2 8 14 Element to find and to delete: -7 8 2 14 -9 6 10 * Element to find and to delete: -9 8 6 14 2 * 10 *	Удалены из дерева узлы со значениями: -1, -3, -5, -7 и -9.
4	f u w 12 4 s 0 l v e 12 4 0		tree = f u w 12 4 s 0 l v e l 4 v 12 f u w 0 * e * s * * * Element to find and to delete: 12 l e v 4 f u w 0 * * * s * * * Element to find and to delete: 4 s f v e l u w 0 * * * * * * * Element to find and to delete: 0 s f v e l u w	Удалены из дерева узлы со значениями: 12, 4 и 0.

5	25 g s 56 h 8 10 we ty g s h we ty		Удалены из дерева узлы со значениями: g, s, h, we и ty.
6	6 5 4 3 2 1		Не были введены узлы для поиска
7	Была выполнена команда ./lab5		Программа просит ввести дерево, затем элементы для поиска