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

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Алгоритмы и структуры данных»
Тема: Программирование алгоритмов с бинарными деревьями
Вариант 1-д

Студент гр. 8304 _____ Николаева М. А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2019

Цель работы.

Познакомиться с такой часто используемой на практике, особенно при решении задач кодирования и поиска, нелинейной структурой данных, как бинарное дерево, способами её представления и реализации, получить навыки решения задач обработки бинарных деревьев.

Постановка задачи.

Вариант 1в: Задано бинарное дерево *b* типа *BT* с типом элементов *Elem*. Для введенной пользователем величины *E* (**var** *E*: *Elem*):

а) определить, входит ли элемент *E* в дерево *b*;

б) определить число вхождений элемента *E* в дерево *b*;

в) найти в дереве *b* длину пути (число ветвей) от корня до ближайшего узла с элементом *E* (если *E* не входит в *b*, за ответ принять −1).

Описание алгоритма.

Для начала программа должна считать данные и занести их в структуру. Дерево реализуется на базе вектора: в структуре каждого узла должен храниться индекс левого и правого узлов в массиве. Если узел в дереве пустой, то массив с данным индексом хранит нулевой элемент.

Далее для нахождения элемента в массиве необходимо сначала идти по левому поддереву, если натыкаемся на нулевой узел, то возвращаемся на шаг назад и проходим по правому поддереву. Для подсчета длины пути до ближайшего искомого элемента необходимо каждый раз, когда переходим в левое или правое поддерево увеличивать длину пути на единицу, когда возвращаемся на шаг назад нужно уменьшать длину пути на единицу. Затем сравнить каждую из получившихся длин.

Спецификация программы.

Программа предназначена для определения вхождения элемента в дерево, подсчета количества искомых элементов, нахождение длины пути до ближайшего элемента.

Программа написана на языке C++. Входными данными являются бинарное дерево и элемент, считываемые из файла или вводимые с клавиатуры. Выходными данными являются промежуточные значения и конечный результат. Данные выводятся на экран монитора.

Описание функций и СД.

```
int minDepthToElem(const T& elem, size_t depth = 0);
```

Метод принимает искомый элемент и возвращает значение типа *int*. Если корень дерева совпадает с искомым элементом, функция возвращает значение глубины. Иначе функция возвращает наибольшее значение рекурсивного применения функции к поддеревьям (если они существуют). Если поддеревьев нет, функция возвращает -1.

```
bool elemIsExist(const T& elem) const;
```

Метод принимает искомый элемент и возвращает значение типа *bool*.

Если корень дерева совпадает с искомым элементом, функция возвращает *true*. Иначе, если у дерева есть левая и правая ветви, функция применяется к поддеревьям рекурсивно. Если ветвей нет, возвращает false.

```
size_t numberOfElem(const T& elem) const;
```

Метод принимает искомый элемент и возвращает значение типа *size_t*. Если корень дерева совпадает с искомым элементом, функция возвращает 1 + (рекурсивное применение функции к левому и правому поддеревьям (если они существуют)).

Тестирование.

Результаты тестирования программы приведены в табл. 1.

Таблица 1 – Результаты тестирования

Ввод	Вывод
(a(b(c(d(d(e(f(d)))))))) d	Элемент найден Количество: 3 Кратчайший путь равен 3
(a(b)(c)) d	Элемент не найден
(a(b(t)(r(u)(d)))(c(k)(d))) d	Элемент найден Количество: 2 Кратчайший путь равен 2
(a(a)(b))	Элемент введен неверно
(a) a	Элемент найден Количество: 1 Кратчайший путь равен 0
() d	Не был добавлен элемент в узел Ошибка ввода
a(b)(c) c	Скобочная запись бинарного дерева должна начинаться и заканчиваться скобками Ошибка ввода
(a(b)(c)(d)) d	Веток больше, чем 2 Ошибка ввода

(a(b)(c) a	Ошибка ввода
(a(b)c)) a	Ошибка! Несколько элементов подряд без скобок

Выводы.

В ходе работы были приобретены навыки работы с бинарными деревьями, изучены обходы бинарных деревьев и реализация деревьев на базе ссылок.

Приложение А.

mybinarytree.h

```
#ifndef MYBINARYTREE_H
#define MYBINARYTREE_H

#include <iostream>
#include <ostream>
#include <memory>


template <class T>
class BinaryTree
{
public:
    typedef std::shared_ptr<BinaryTree<T>> BinaryTreeP;

    explicit BinaryTree() = default;
    explicit BinaryTree(const T& data);
    ~BinaryTree() = default;

    BinaryTreeP getLeft() const;
    BinaryTreeP getRight() const;
    const T& getData() const;

    static BinaryTreeP createCharTree(const std::string& expression) = delete;

    static size_t getEndIndexSubTree(const std::string& str, size_t indexEnd);

    bool elemIsExist(const T& elem) const;
    size_t numberOfElem(const T& elem) const;
    int minDepthToElem(const T& elem, size_t depth = 0);

private:
    BinaryTreeP leftTree_;
    BinaryTreeP rightTree_;
    T data_;
};


template<class T>
BinaryTree<T>::BinaryTree(const T &data)
{
    data_ = data;
}


template<class T>
typename BinaryTree<T>::BinaryTreeP BinaryTree<T>::getLeft() const
{
    return this->leftTree_;
}
```

```
template<class T>
typename BinaryTree<T>::BinaryTreeP BinaryTree<T>::getRight() const
{
    return this->rightTree_;
}


template<class T>
const T &BinaryTree<T>::getData() const
{
    return this->data_;
}


template<>
typename BinaryTree<char>::BinaryTreeP
BinaryTree<char>::createCharTree(const std::string& str)
{
    BinaryTreeP tree = std::make_shared<BinaryTree<char>>();

    char elem = 0;
    size_t indexStart = 1;

    while (str[indexStart] != '(' && str[indexStart] != ')') {
        if (str[indexStart] != ' ' && str[indexStart] != '(' && str[indexStart] != ')')
{
            elem = str[indexStart];
            tree->data_ = elem;
        }
        indexStart++;;
    }

    if (str[indexStart] == ')') {
        return tree;
    }

    size_t indexEnd = indexStart + 1;
    indexEnd = getEndIndexSubTree(str, indexEnd);
    BinaryTreeP  leftTree  =  createCharTree(str.substr(indexStart,  indexEnd  -  index-
Start));
    tree->leftTree_ = leftTree;

    indexStart = indexEnd;
    while (str[indexStart] != '(' && str[indexStart] != ')') {
        indexStart++;
    }

    if (str[indexStart] == ')') {
        return tree;
    }

    indexEnd = indexStart + 1;
```

```
    indexEnd = getEndIndexSubTree(str, indexEnd);
    BinaryTreeP rightTree = createCharTree(str.substr(indexStart, indexEnd - index-
Start));
    tree->rightTree_ = rightTree;

    return tree;
}


template<class T>
size_t BinaryTree<T>::getEndIndexSubTree(const std::string &str, size_t indexEnd)
{
    /*
     * Метод возвращает индекс в строке конца поддерева
    */

    size_t openB = 1;
    size_t closeB = 0;
    while (openB != closeB) {
        if (str[indexEnd] == '(') {
            openB++;
        }
        else if (str[indexEnd] == ')') {
            closeB++;
        }
        indexEnd++;
    }
    return indexEnd;
}


template<class T>
bool BinaryTree<T>::elemIsExist(const T& elem) const
{
    if (this->getData() == elem)
        return true;

    bool isExist = false;
    if (this->getLeft() != nullptr) {
        isExist = isExist || this->getLeft()->elemIsExist(elem);
    }

    if (this->getRight() != nullptr) {
        isExist = isExist || this->getRight()->elemIsExist(elem);
    }

    return isExist;
}


template<class T>
size_t BinaryTree<T>::numberOfElem(const T& elem) const
{
```

```
    size_t count = 0;
    if (this->getData() == elem) {
        count += 1;
    }

    if (this->getLeft() != nullptr) {
        count += this->getLeft()->numberOfElem(elem);
    }

    if (this->getRight() != nullptr){
        count += this->getRight()->numberOfElem(elem);
    }

    return count;
}

template<class T>
int BinaryTree<T>::minDepthToElem(const T &elem, size_t depth)
{
    if (this->getData() == elem){
        return static_cast<int>(depth);
    }

    int leftMinDepth = -1;
    int rightMinDepth = -1;

    if (this->getLeft() != nullptr) {
        leftMinDepth = this->getLeft()->minDepthToElem(elem, depth + 1);
    }

    if (this->getRight() != nullptr){
        rightMinDepth = this->getRight()->minDepthToElem(elem, depth + 1);
    }

    if (leftMinDepth != -1 && rightMinDepth != -1)
        return std::min(leftMinDepth, rightMinDepth);
    else if (leftMinDepth == -1)
        return rightMinDepth;
    else
        return leftMinDepth;

}


#endif // MYBINARYTREE_H
```

main.cpp

```
#include <iostream>
#include <string>
#include <fstream>
#include "mybinarytree.h"
```

```
using std::cout;
using std::cin;
using std::cerr;


std::string readStr(std::istream& in);
std::string readStr(std::ifstream& in);

bool isCorrectStr(const std::string& str);


int main(int argc, char *argv[]) {
    setlocale(LC_ALL, "");

    if (argc < 3) {
        std::string expression;
        std::string result;
        std::string element;
        char elem;

        if (argc == 2) {
            std::string fileName = argv[1];
            std::ifstream inputFile (fileName, std::ios::in);
            if (!inputFile.is_open()) {
                cerr << "Error. I ncorrect file name!\n";
                return 0;
            }
            expression = readStr(inputFile);
            element = readStr(inputFile);
            inputFile.close();
        }
        else {
            cout << "Enter a tree:\n";
            expression = readStr(cin);

            cout << "Enter an element:\n";
            element = readStr(cin);
        }

        result += "Input expression:\n" + expression + "\nInput element:\n" + element +
"\n";

        bool isCorrectInput = true;

        if (element.length() > 1){
            result += "Incorrect element\n";
            isCorrectInput = false;
        };

        if (!isCorrectStr(expression)) {
            result += "Incorrect expression\n";
            isCorrectInput = false;
```

```
        }

        if (isCorrectInput) {
            elem = *(element.c_str());

            BinaryTree<char>::BinaryTreeP            tree            =            Bina-
ryTree<char>::createCharTree(expression);

            if (tree->elemIsExist(elem)){
                result += "\nThe elemnt \"" + element + "\" exists in the tree.\n";
            }
            else {
                result  +=  "\nThe  elemnt  \""  +  element  +  "\"  doesnt  exist  in  the
tree.\n";
            }

            size_t count = tree->numberOfElem(elem);
            result += "The number of elements \"" + element +
                    "\" is " + std::to_string(count) + "\n";

            int depth = tree->minDepthToElem(elem);
            result += "The length of the minimum path to the element \"" + element +
                    "\" is " + std::to_string(depth) + "\n";
        }

       result += "\n\n\n";
        cout << result;

        std::ofstream outputFile("result.txt", std::ios::app);
        outputFile << result;
        outputFile.close();
    }
    else {
        cerr << "Error: incorect console's arguments\n";
    }

    return 0;
}


std::string readStr(std::istream& in) {
    std::string result;
    std::string str;
    getline(in, str);
    result += str;
    result += "\n";

    result = result.substr(0, result.size() - 1);

    return result;
}

std::string readStr(std::ifstream& in) {
```

```
    std::string result;
    std::string str;
    getline(in, str);
    result += str;
    result += "\n";

    result = result.substr(0, result.size() - 1);

    return result;
}


bool isCorrectStr(const std::string &str)
{

    bool isElem = false;
    bool flagIsCorrect = true;
    if (str[0] != '(' || str[str.length() - 1] != ')') {
        cout << "This binary tree representation must begin and end with brackets\n";
        return false;
    }

    size_t indexStart;
    size_t numberOfBrackets = 0;
    for (indexStart = 1; indexStart < str.length() - 1; indexStart++) {
        if (str[indexStart] != ' ' && str[indexStart] != '(' && str[indexStart] != ')')
{
            if (!isElem) {
                isElem = true;
            }
            else {
                cout << "Multiple items in a row without parentheses";
                return false;
            }
        }
        if (str[indexStart] == '(') {
            if (!isElem) {
                cout << "An item was not added to the node";
                return false;
            }
            numberOfBrackets++;
            if (numberOfBrackets > 2) {
                cout << "An item was not added to the node.";
                return false;
            }
            size_t indexEnd = indexStart;
            int openB = 1;
            int closeB = 0;
            while (openB > closeB) {
                indexEnd++;
                if (indexEnd >= str.length()) {
                    cout << "Incorrect enter string";
                    return false;
```

```
                }
                if (str[indexEnd] == '(') {
                    openB++;
                }
                else if (str[indexEnd] == ')') {
                    closeB++;
                }
            }
            flagIsCorrect = flagIsCorrect && isCorrectStr(str.substr(indexStart, index-
End - indexStart+ 1));
            indexStart= indexEnd;
        }
    }
    if (str[indexStart] == ')') {
        if (!isElem) {
            cout << "An item was not added to the node";
            return false;
        }
        if (indexStart== str.length() - 1) {
            return flagIsCorrect;
        }
        else {
            cout << "Incorrect entering";
            return false;
        }
    }
    return false;
}
```