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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Алгоритмы и структуры данных»

Тема: Бинарные деревья

Студент гр. 8304 _____ Алтухов А.Д.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2019

Цель работы.

Ознакомиться с линейными структурами данных, обеспечивающими доступ к данным только через начало и/или конец структуры, способами их реализации.

Задание.

Вариант 2-в.

Для заданного бинарного дерева b типа BT с произвольным типом элементов:

- определить максимальную глубину дерева b, т. е. число ветвей в самом длинном из путей от корня дерева до листьев;

- вычислить длину внутреннего пути дерева b, т. е. сумму по всем узлам длин путей от корня до узла;

- напечатать элементы из всех листьев дерева b;

- подсчитать число узлов на заданном уровне n дерева b (корень считать узлом 1-го уровня);

Описание алгоритма.

Определение максимальной глубины: рекурсивно вызывается функция подсчета глубины для левого и правого поддеревьев, на каждом уровне к текущей высоте добавляется единица. Учитывается только один путь до листа — максимальный.

Вычисление длины внутреннего пути: если существуют поддеревья, то к счетчику добавляется числовое значение уровня дерева, который сейчас обрабатывается алгоритмом. После этого функция рекурсивно вызывается для поддерева с увеличением уровня.

Печать элементов из листьев: если поддеревьев нет, то печатаем элемент. В обратном случае рекурсивно вызываем функцию для существующих поддеревьев.

Подсчет узлов на заданном уровне дерева: в случае, если уровень узла соответствует искомому, возвращаем единицу, иначе рекурсивно вызываем функцию для поддеревьев и складываем возвращенные результаты.

Основные функции и структуры.

`class binaryTree` — класс, содержащий в себе информацию о бинарном дереве. Присутствуют следующие поля:

`Elem<T>* arr` — содержит все элементы дерева.
`int curIndex` — номер обрабатываемого элемент.
`int* infoArr` — информация о дереве — занимаемая память, следующая свободная ячейка.
`bool isMainTree` — является ли объект основным деревом или поддеревом.
`struct Elem` — структура, содержащая элемент дерева.
`T value` — значение узла.
`int left` — индекс левого поддерева в массиве arr.
`int right` — индекс правого поддерева в массиве arr.
Существуют следующие методы для взаимодействия с бинарным деревом:
`bool isEmptyTree() const;`
`bool isEmptyElem() const` — проверяет, существует ли дерево/элемент.
`T getRoot() const` — возвращает значение текущего узла.
`binaryTree<T> getLeftTree();`
`binaryTree<T> getRightTree()` — возвращает левое/правое поддерево.
`void setLeft(T newValue);`
`void setRight(T newValue);`
`void setRoot(T newValue)` — устанавливает новое значение в узел слева/справа/текущий узел.
`bool readTree(std::ifstream& inputStream)` — преобразует строковое представление дерева в необходимую для дальнейшей обработки форму.
`int calcHeight();`
`int pathLength(int level=1);`
`void printLeaves();`
`int countNodesOnLevel(int level)` — функции, выполняющие соответствующие заданию подзадачи. Их описание расположено выше.

Тестирование.

Некорректные данные:

№	Ввод	Ожидаемый ответ	Полученный ответ
1	12345	Сообщение об ошибке	Нет открывающей скобки
2	(1#	Сообщение об ошибке	Лишний #
3	2)	Сообщение об ошибке	Нет открывающей скобки

Корректные данные:

№	Ввод	Ожидаемый ответ	Полученный ответ
4	(1)	Высота дерева: 1 Длина внутреннего пути: 0 Список листьев: 1 Узлов на уровне 3: 0	Высота дерева: 1 Длина внутреннего пути: 0 Список листьев: 1 Узлов на уровне 3: 0
5	(1 (2)(3))	Высота дерева: 2 Длина внутреннего пути: 2 Список листьев: 2 3 Узлов на уровне 3: 0	Высота дерева: 2 Длина внутреннего пути: 2 Список листьев: 2 3 Узлов на уровне 3: 0
6	(1 (2 (3 (4)(5))))	Высота дерева: 4 Длина внутреннего пути: 9 Список листьев: 4 5 Узлов на уровне 3: 1	Высота дерева: 4 Длина внутреннего пути: 9 Список листьев: 4 5 Узлов на уровне 3: 1
7	(1 (2 (3 (5))(4))(6))	Высота дерева: 4 Длина внутреннего пути: 9 Список листьев: 5 4 6 Узлов на уровне 3: 2	Высота дерева: 4 Длина внутреннего пути: 9 Список листьев: 5 4 6 Узлов на уровне 3: 2
8	(1 (2 (3 (5))(4))(6 (7) (8)))	Высота дерева: 4 Длина внутреннего пути: 13 Список листьев: 5 4 7 8 Узлов на уровне 3: 4	Высота дерева: 4 Длина внутреннего пути: 13 Список листьев: 5 4 7 8 Узлов на уровне 3: 4
9	(1 (2 (3 (5))(4))(6 #(8)))	Высота дерева: 4 Длина внутреннего пути: 11	Высота дерева: 4 Длина внутреннего пути: 11

		Список листьев: 5 4 8 Узлов на уровне 3: 3	Список листьев: 5 4 8 Узлов на уровне 3: 3

Выводы.

В ходе выполнения лабораторной работы была исследована такая структура данных, как бинарное дерево, изучены методы работы с ней. Были реализованы некоторые алгоритмы для взаимодействия с деревом.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД

Файл binaryTree.h

```
#define SIZE 100
#include <iostream>
#include <fstream>
#include <string>

enum infoIndex {realArrSize, maxArrSize}; //сколько занято памяти и
сколько выделено

template <typename T>
struct Elem{
    T value;
    int left;
    int right;
    Elem():left(-1), right(-1){}
    Elem(T value):value(value), left(-1), right(-1){}
};

template <typename T>

class BinaryTree{


    Elem<T>* arr;
    int curIndex;
    int* infoArr;
    bool isMainTree;

    void resize();

public:
    BinaryTree(): arr(nullptr), curIndex(-1), infoArr(new int[2]),
isMainTree(true){
        infoArr[realArrSize] = 0;
        infoArr[maxArrSize] = 0;
    }
    BinaryTree(Elem<T>*& arr, int curIndex, int*& infoArr): arr(arr),
curIndex(curIndex), infoArr(infoArr), isMainTree(false){}
    ~BinaryTree();
    bool isEmptyTree() const;
    bool isEmptyElem() const;
    Elem<T> getLeftElem() const;
    Elem<T> getRightElem() const;
    T getRoot() const;
    BinaryTree<T> getLeftTree();
    BinaryTree<T> getRightTree();
    void setLeft(); //инициализация без установки значения
    void setRight();
    void setLeft(T newValue);
```

```
    void setRight(T newValue);
    void setRoot(T newValue);
    void printTree();
    bool readTree(std::ifstream& inputStream);
    int calcHeight();
    int pathLength(int level=1);
    void printLeaves(std::ofstream& outputF);
    int countNodesOnLevel(int level);
};

template <typename T>
BinaryTree<T>::~BinaryTree(){
    if (isMainTree && arr){ //чтобы поддерево случайно не удалило все
        delete[] arr;
        delete[] infoArr;
        arr = nullptr;
        infoArr = nullptr;
    }

}

template <typename T>
void BinaryTree<T>::resize(){

    Elem<T>* tempArr = new Elem<T>[infoArr[maxArrSize]+SIZE];
    for (int i=0; i<infoArr[realArrSize]; i++)
        tempArr[i] = arr[i];
    delete[] arr;
    arr = tempArr;
}

template <typename T>
bool BinaryTree<T>::isEmptyTree() const {

    return !infoArr[realArrSize]; //инициализировано ли дерево
}

template <typename T>
bool BinaryTree<T>::isEmptyElem() const {

    return (curIndex == -1); //есть ли что-нибудь по этому индексу
}
template <typename T>
Elem<T> BinaryTree<T>::getLeftElem() const {

    if (!isEmptyElem()){
        return arr[arr[curIndex].left];
    }
    else {
        //пустой элем
        std::cout << "Пустой элемент был возвращен!\n";
```

```
        return Elem<T>();
    }
}

template <typename T>
Elem<T> BinaryTree<T>::getRightElem() const {

    if (!isEmptyElem()){
        return arr[arr[curIndex].right];
    }
    else {
        //пустой элем
        std::cout << "Пустой элемент был возвращен!\n";
        return Elem<T>();
    }
}

template <typename T>
T BinaryTree<T>::getRoot() const {

    if (!isEmptyElem()){
        return arr[curIndex].value;
    }
    else {
        //пустой элем
        std::cout << "Пустой элемент был возвращен!\n";
        return 0;
    }
}

template <typename T>
BinaryTree<T> BinaryTree<T>::getLeftTree() {

    if ((!isEmptyElem()) && (arr[curIndex].left != -1)){
        return BinaryTree<T>(arr, arr[curIndex].left, infoArr);
    }
    else {
        std::cout << "Левое поддерево не существует\n";
        return BinaryTree<T>();
    }
}

template <typename T>
BinaryTree<T> BinaryTree<T>::getRightTree() {

    if ((!isEmptyElem()) && (arr[curIndex].right != -1)){
        return BinaryTree<T>(arr, arr[curIndex].right, infoArr);
    }
    else {
        std::cout << "Правое поддерево не существует\n";
        return BinaryTree<T>();
```

```
    }
}


template <typename T>
void BinaryTree<T>::setLeft(){

    if (infoArr[realArrSize]==infoArr[maxArrSize]){
        this->resize();
    }
    if (!isEmptyElem()){
        arr[curIndex].left = infoArr[realArrSize];
        infoArr[realArrSize]++;
    }

}

template <typename T>
void BinaryTree<T>::setLeft(T newValue){

    if (infoArr[realArrSize]==infoArr[maxArrSize]){
        this->resize();
    }
    if (!isEmptyElem()){
        arr[curIndex].left = infoArr[realArrSize];
        arr[infoArr[realArrSize]] = Elem<T>(newValue);
        infoArr[realArrSize]++;
    }

}

template <typename T>
void BinaryTree<T>::setRight(){

    if (infoArr[realArrSize]==infoArr[maxArrSize]){
        this->resize();
    }
    if (!isEmptyElem()){
        arr[curIndex].right = infoArr[realArrSize];
        infoArr[realArrSize]++;
    }
}


template <typename T>
void BinaryTree<T>::setRight(T newValue){

    if (infoArr[realArrSize]==infoArr[maxArrSize]){
        this->resize();
    }
    if (!isEmptyElem()){
```

```
        arr[curIndex].right = infoArr[realArrSize];
        arr[infoArr[realArrSize]] = Elem<T>(newValue);
        infoArr[realArrSize]++;
    }
}

template <typename T>
void BinaryTree<T>::setRoot(T newValue){
    if (isEmptyTree()){
        arr = new Elem<T>[SIZE];
        infoArr[maxArrSize] = SIZE;
        infoArr[realArrSize] = 1;
        curIndex = 0;
    }
    if (!isEmptyElem()){
        arr[curIndex].value = newValue;
    }
}


void destroySpaces(std::ifstream& inputStream){
    char c ='\0';
    while ((inputStream >> c) && (c==' ')){}
    if (c!=' ')
        inputStream.unget();
}

template <typename T>
bool BinaryTree<T>::readTree(std::ifstream& inputStream){

    destroySpaces(inputStream);
    char c = '\0';
    if ((inputStream >> c) && (c=='(')){ //считывание узла

        destroySpaces(inputStream);
        T inputVal;

        if (inputStream>>inputVal){

            this->setRoot(inputVal);
            destroySpaces(inputStream);

            if ((inputStream >> c) && (c=='(')){ //считывание левого
поддерева

                inputStream.unget();
                this->setLeft();

                if (this->getLeftTree().readTree(inputStream)){}
                else return false;
```

```
                if ((inputStream >> c) && (c=='('){ //считывание
правого поддерева (если есть)
                    inputStream.unget();
                    this->setRight();
                    if (this->getRightTree().readTree(inputStream)){}
                }
                else
                    inputStream.unget();
            }
            else if (c=='#') { //если нет скобки, то может левое
поддерево отсутствует?
                destroySpaces(inputStream);
                if ((inputStream >> c) && (c=='('){ //видимо да,
считываем правое (если есть)
                    inputStream.unget();
                    this->setRight();
                    if (this->getRightTree().readTree(inputStream)){}
                    else return false;
                }
                else {
                    std::cout << "Лишний #\n";
                    return false;
                }
            }
            else {
                inputStream.unget();
            }

        }
        else {
            std::cout << "Нет значения узла\n";
            return false;
        }
    }
    else {
        std::cout << "Нет открывающей скобки\n";
        return false;
    }
    destroySpaces(inputStream);
    if ((inputStream >> c) && (c==')')){}
    else {
        std::cout << "Нет закрывающей скобки\n";
        return false;
    }

    return true;

}

template <typename T>
void BinaryTree<T>::printTree(){
```

```
    if (!isEmptyElem()){
        std::cout << this->getRoot() << " ";
        if (this->arr[curIndex].left != -1)
            this->getLeftTree().printTree();
        if (this->arr[curIndex].right != -1)
            this->getRightTree().printTree();
    }
}

template <typename T>
int BinaryTree<T>::calcHeight(){

    if (isEmptyElem()){
        return 0;
    }
    return std::max((this->arr[curIndex].left != -1) ? this-
>getLeftTree().calcHeight() : 0,  (this->arr[curIndex].right != -1) ?
this->getRightTree().calcHeight():0) + 1;
}

template <typename T>
int BinaryTree<T>::pathLength(int level){

    int counter = 0;

    if (isEmptyElem())
        return 0;
    if (this->arr[curIndex].left != -1){
        counter += level;
        counter += this->getLeftTree().pathLength(level+1);
    }
    if (this->arr[curIndex].right != -1){
        counter += level;
        counter += this->getRightTree().pathLength(level+1);
    }
    return counter;
}

template <typename T>
void BinaryTree<T>::printLeaves(std::ofstream& outputF){

    if (isEmptyElem())
        return;
    if ((this->arr[curIndex].left == -1) && (this->arr[curIndex].right
== -1)){
        std::cout << this->getRoot() << " ";
              outputF << this->getRoot() << " ";
       }
    else {
        if (this->arr[curIndex].left != -1)
```

```
            this->getLeftTree().printLeaves(outputF);
        if (this->arr[curIndex].right != -1)
            this->getRightTree().printLeaves(outputF);
    }
}

template <typename T>
int BinaryTree<T>::countNodesOnLevel(int level){

    level--;
    if (isEmptyElem()){
        return 0;
       }
       //std::cout << "Обработка уровня " << level+1 << ". Значение
узла: " << this->getRoot() <<((level==0) ? ". +1 к счетчику.\n" :
".\n");
    if (level == 0){
        return 1;
    }
    return ((this->arr[curIndex].left != -1) ? this-
>getLeftTree().countNodesOnLevel(level) : 0) + ((this-
>arr[curIndex].right != -1) ? this-
>getRightTree().countNodesOnLevel(level) : 0);

}
```

Файл main.cpp

```
#include <iostream>
#include <string>
#include <fstream>
#include <locale>
#include "binaryTree.h"


int main(int argc, char *argv[])
{
       std::cout << "\n\n=====Запуск программы=====\n";
    setlocale(LC_ALL, "RU");
    std::ifstream inputF;

       if (argc > 1)
        inputF.open(argv[1]);
    else
        inputF.open("input.txt");

    if (!inputF.is_open()) {
        std::cerr << "Невозможно открыть файл со входными данными";
        return 0;
    }
```

```
    std::ofstream outputF("output.txt");
    if (!outputF.is_open()) {
        std::cerr << "Невозможно открыть файл вывода";
        return 0;
    }

    BinaryTree<int> bt;
    bool readRes = bt.readTree(inputF);
       if (!readRes)
              return 0;

    std::cout << "Высота дерева: " << bt.calcHeight()<<"\n";
       std::cout << "Длина внутреннего пути: "
<<bt.pathLength()<<"\n";
       std::cout << "Список листьев: ";
       outputF << "Список листьев: ";
       outputF << "\n";
       bt.printLeaves(outputF);
       std::cout << "\n";
    std::cout << "Узлов на уровне 3: " << bt.countNodesOnLevel(3) <<
"\n";


       outputF << "Высота дерева: " << bt.calcHeight()<<"\n";
       outputF << "Длина внутреннего пути: " <<bt.pathLength()<<"\n";
    outputF << "\nУзлов на уровне 3: " << bt.countNodesOnLevel(3) <<
"\n";

       std::cout << "\n\n=====Завершение программы=====\n\n";

    inputF.close();
    outputF.close();

    return 0;
}
```