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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Алгоритмы и структуры данных»

Тема: Деревья

Студент гр. 8381 _____ Муковский Д.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2019

Цель работы.

Ознакомиться с основными характеристиками и особенностями такой структуры данных, как бинарное дерево, изучить особенности ее реализации на языке программирования *C++*. Разработать программу, выводящее бинарное дерево по уровням.

Задание.

Задано бинарное дерево *b* типа *BT* с произвольным типом элементов. Используя очередь и операции над ней, напечатать все элементы дерева b по уровням: сначала – из корней дерева, затем (слева направо) – из узлов, сыновьих по отношению к корню, затем (также слева направо) – из узлов, сыновьих по отношению к этим узлам и т.д.

Основные теоретические положения.

Дерево – конечное множество *T*, состоящее из одного или более узлов, таких, что:

а) имеется один специально обозначенный узел, называемый корнем данного дерева;

б) остальные узлы (исключая корень) содержатся в m ³ 0 попарно не пересекающихся множествах *T*1, *T*2, ..., *Tm*, каждое из которых, в свою очередь, является деревом. Деревья *T*1, *T*2, ..., *Tm* называются поддеревьями данного дерева.

При программировании и разработке вычислительных алгоритмов удобно использовать именно такое рекурсивное определение, поскольку рекурсивность является естественной характеристикой этой структуры данных.

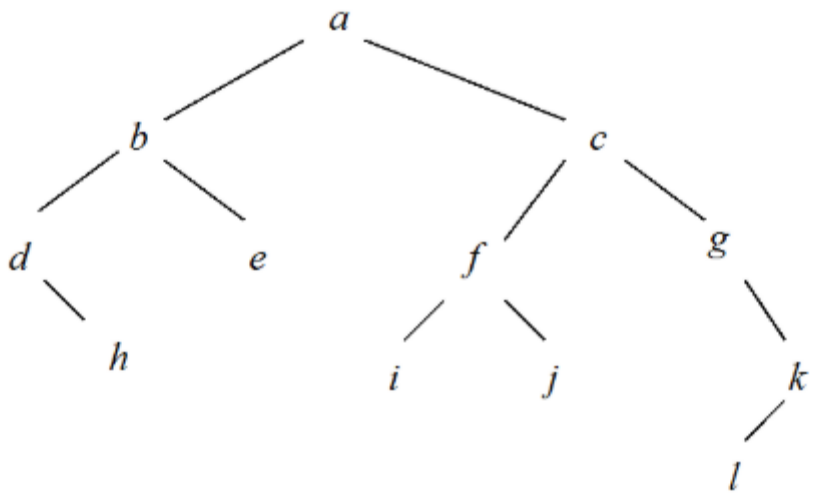


Рисунок 1 - Бинарное дерево

Бинарное дерево - конечное множество узлов, которое либо пусто, либо состоит из корня и двух непересекающихся бинарных деревьев, называемых правым поддеревом и левым поддеревом

Выполнение работы.

Написание работы производилось в среде разработки *QtCreator* с использованием фреймворка *Qt*. Сборка, отладка производились в *QtCreator*. Исходные коды файлов программы представлены в приложениях А-Ж.

Для реализации программы был разработан графический интерфейс с помощью встроенного в *QtCreator UI*-редактора. Он представляет из себя поле ввода, кнопку считывания из файла, а также кнопку запуска обработчика со своим графическим интерфейсом. В свою очередь он представляет из себя поле с заданным деревом, сцену, на которой дерево изображается, *comboBox* с выбором параметров запуска, а также кнопку сохранения результата. Основные слоты для работы графического интерфейса приведены в табл. 1.

Таблица 1 – Слоты класса *MainWindow* и их назначение

Метод	Назначение
`on_readFileButton_clicked()`	Слот, отвечающий за считывание данных из файла
`on_pushButton_clicked()`	Слот, отвечающий за запуск обработчика

Для реализации бинарного дерева были созданы структура узла *Node*, в котором хранится информация и указатели на правого и левого потомка, а также был создан класс *BinTree* который хранит корень дерева. Были реализованы методы данного класса, приведенные в табл. 2.

Таблица 2 – Основные методы работы с классом бинарного дерева

Метод	Назначение
`void BinTree()`	Конструктор бинарного дерева
`Node* createTree(QStringList, int& numb)`	Создает бинарное дерево из массива строк-элементов, полученного из входной строки
`int getDepth(Node *tree)`	Возвращает глубину дерева
`int getNumbElem()`	Возвращает количество узлов дерева

Программа имеет возможность графического отображения полученного бинарного дерева с помощью виджета *QGraphicsView*. Функции, необходимые для графического представления дерева, представлены в табл. 3.

Таблица 3 –Функции, связующие графический интерфейс и алгоритмы

Метод	Назначение
`QGraphicsScene *graphicConfig (BinTree *tree, QGraphicsScene *&scene, int depth)`	По заданному бинарному дереву выполняет рисование в объекте *QGraphicsScene*
`int netTreePainter (QGraphicsScene *&scene, Node *node, int w, int h, int wDelta, int hDelta, QPen &pen, QBrush &brush, QFont &font, int depth)`	Рекурсивный алгоритм обхода дерева и рисования узлов в заданном объекте *QGraphicsScene*

Также было реализовано модальное диалоговое окно *CustomOutput*, в которое посредством сигнала *sendArray*(), реализованного в классе *MainWindow*, посылалась строка содержащая скобочную запись дерева, в свою очередь принималась она с помощью слота *recieveArray*(). После чего полученная строка выводилась в поле *input*(*TextBrowser*) класса *CustomOutput*.

Далее пользователь мог выбрать каким образом он хочет запустить программу пошагово или нет. В случае пошагового выполнения появлялась кнопка “Следующий шаг”, которая выполняла ровно один шаг алгоритма, иначе алгоритм исполнялся полностью. После завершения работы алгоритма, появлялась кнопка “Сохранить”, с помощью которой пользователь мог выбрать любой текстовый файл из директории и сохранить полученный результат.

Все методы класса *CustomOutput* приведены в табл. 4.

Таблица 4 – Методы класса *CustomOutput*

Метод	Назначение
on_pushButton_clicked()	Запуск алгоритма в выбранном режиме
on_nextStep_clicked()	В случая пошагового режима переход к следующему шагу
on_saveButton_clicked()	Сохранение результата

За основу главного алгоритма вывода дерева по уровням был взят алгоритм поиска в ширину. Описание алгоритма:

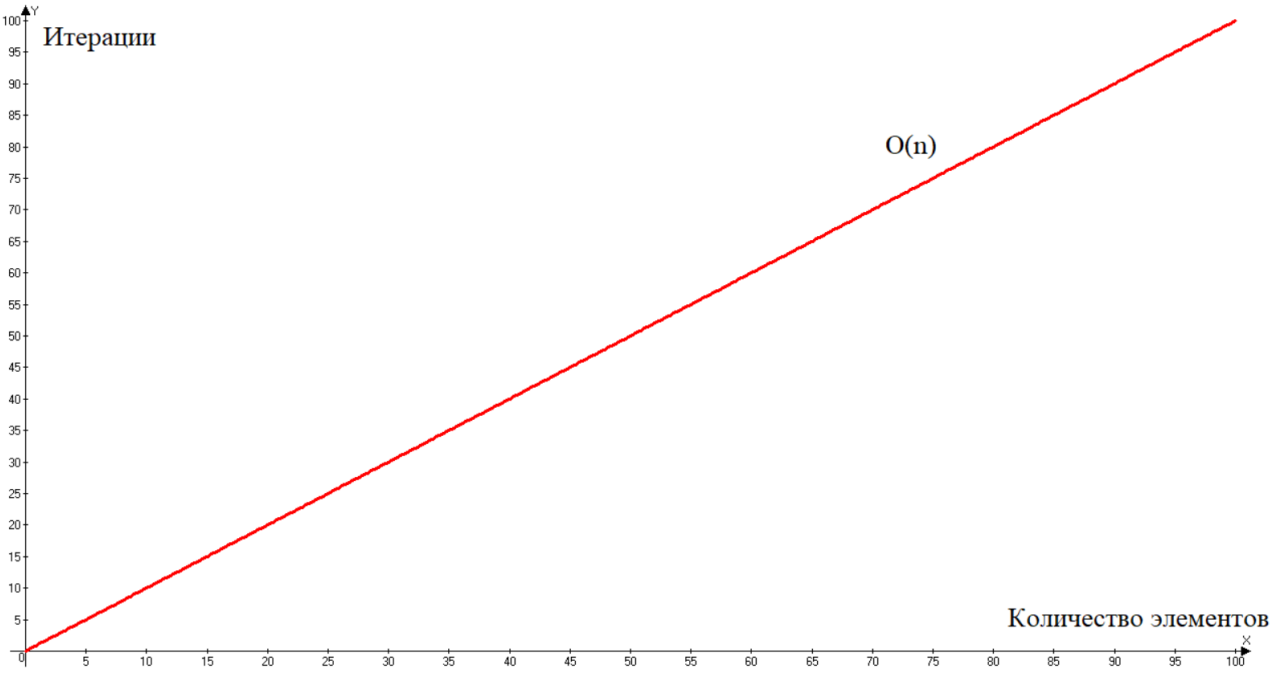
1. Поместить узел, с которого начинается поиск, в изначально пустую очередь.
2. Извлечь из начала очереди узел и пометить его как развёрнутый.
    - Если узел является целевым узлом и очередь пуста, то завершить поиск с результатом «успех».

- В противном случае, в конец очереди добавляются все преемники узла , которые ещё не развёрнуты и не находятся в очереди.

3. В случае, если уровень текущего элемента увеличился добавить в вывод перенос строки
4. Вернуться к п. 2.

Оценка сложности алгоритма.

Алгоритм вывода всех узлов дерева по уровням являются рекурсивными, каждый узел дерева обрабатывается один раз, следовательно, сложность алгоритма $O(N)$.



Тестирование программы.

Был проведен ряд тестов, проверяющих корректность работы программы. Результаты тестирование приведены в табл. 4.

Таблица 4 – Тестирование программы

Входная строка	Вывод
(a(d(t)(y))(t(r)(j (i))))	Level:1 a Level:2 d t Level:3 t y r j Level:4 i
(4 (5) (5	Некорректная запись дерева
(6 (4 (5) (9)) (5))	Level:1 6 Level:2 4 5 Level:3 5 9
(6 (4 (5) (9)) (5 (7) (9)))	Level:1 6 Level:2 4 5 Level:3 5 9 7 9

Выводы.

В ходе выполнения лабораторной работы была написана программа, создающая бинарное дерево и выводящая его по уровням. Также был реализован пошаговый вариант алгоритма, позволяющий наглядно увидеть его работу. Печать бинарного дерева выполняется графически.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. MAIN.C

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    bool flag = false;
    for(int i = 1; i < argc; i++)
    {
        if(!strcmp("-console", argv[i]))
            flag = true;
    }
    if(flag){
        consoleMain();
        return 0;
    }
    else
    {
        QApplication a(argc, argv);
        MainWindow w;
        w.show();
        return a.exec();
    }
    return 0;
```

}ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ. MAINWINDOW.H

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "customoutput.h"
#include <QFileDialog>
#include <fstream>
#include <sstream>
#include <iostream>
#include "bracketbalance.h"
#include "console.h"

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
signals:
    void sendArray(QString str);

private slots:
    void on_pushButton_clicked();

    void on_readFileButton_clicked();

private:
    Ui::MainWindow *ui;
    customOutput *cusOuput;
};
      #endif // MAINWINDOW_H
```

ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД ПРОГРАММЫ. MAINWINDOW.CPP

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "isincorrectsymbols.h"
#include <QMessageBox>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    this->setWindowTitle("Вывод дерева по уровням");
}

MainWindow::~MainWindow()
{
    delete ui;
}


void MainWindow::on_pushButton_clicked()
{
    QString inputTree_ = ui->inTree->text();
    QString inputTree = deleteSpaces(inputTree_);
    if (inputTree.isEmpty()){
        QMessageBox::critical(this,"Ошибка","Вы не задали дерево");
        return;
    }
    if (!isIncorrectSymbols(inputTree)){
        QMessageBox::critical(this,"Ошибка","Некорректная запись дерева");
        return;
    }
    if (!bracketBalance(inputTree)){
        QMessageBox::critical(this,"Ошибка","Некорректная запись дерева");
        return;
    }

    cusOuput = new customOutput();
    connect(this, SIGNAL(sendArray(QString)), cusOuput, SLOT(recieveArray(QString)));
    emit sendArray(inputTree);
    cusOuput->exec();
}

void MainWindow::on_readFileButton_clicked()
{
    QString fileName = QFileDialog::getOpenFileName(this, tr("load"), QDir::homePath(),
tr("*.txt"));

        if (QString::compare(fileName, QString()) != 0)
        {
            std::ifstream f(qPrintable(fileName), std::ios::in);
            std::string out;
            getline(f, out);
            f.close();
            ui->inTree->setText(QString::fromStdString(out));
        }
```

```
}
```

ПРИЛОЖЕНИЕ Г

ИСХОДНЫЙ КОД ПРОГРАММЫ.CUSTOMOUTPUT.H

```
#ifndef CUSTOMOUTPUT_H
#define CUSTOMOUTPUT_H

#include <QDialog>

#include "graphicrender.h"
#include "nodequeque.h"
#include <QFileDialog>
#include <fstream>
#include <sstream>
#include <iostream>
namespace Ui {
class customOutput;
}

class customOutput : public QDialog
{
    Q_OBJECT

public:
    explicit customOutput(QWidget *parent = nullptr);
    ~customOutput();
private slots:
    void recieveArray(QString inputArray);
    void on_pushButton_clicked();

    void on_nextStep_clicked();

    void on_saveButton_clicked();

private:
    Ui::customOutput *ui;
    QString inputTreeString;
    QGraphicsScene *scene;
    int lev= 0;
    bool flagStep;
    QString out;
    BinaryTree* BT;
    NodeQueue* queque;
    int depth;
};

#endif // CUSTOMOUTPUT_H
```

ПРИЛОЖЕНИЕ Д

ИСХОДНЫЙ КОД ПРОГРАММЫ. CUSTOMOUTPUT.CPP

```
#include "customoutput.h"
#include "ui_customoutput.h"
#include "stepbystep.h"
customOutput::customOutput(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::customOutput)
{
    ui->setupUi(this);
    scene = new QGraphicsScene;
    ui->graphicsView->setScene(scene);
    this->setWindowTitle("Вывод дерева по уровням");
    ui->comboBox->addItem("Сразу результат");
    ui->comboBox->addItem("Пошагово");
    BT = new(BinaryTree);
    ui->nextStep->hide();
    ui->saveButton->hide();
}

customOutput::~customOutput()
{
    delete ui;
}
void customOutput::recieveArray(QString inputTree){
    ui->input->setText(inputTree);
    inputTreeString = inputTree;
}


void customOutput::on_pushButton_clicked()

{
    lev = 0;
    scene->clear();
    out.clear();
    ui->answer->clear();
    int numb = 1;


    BT->root = BT->createTree(splitForTree(inputTreeString),numb);



    flagStep = false;
    queque = new NodeQueue(numb);

    Node* currentNode = BT->root;
    queque->addItem(currentNode);
    BT->root->level =1;
    depth = BT->getDepth(BT->root);

    int value = ui->comboBox->currentIndex();
    if (!value){
        stepByStepAlg(lev,out,queque,flagStep);
        ui->answer->setText(out);
        graphicConfig(BT, scene,depth);
```

```
        ui->saveButton->show();
    }
    else{
        flagStep =true;
        ui->pushButton->hide();
        ui->comboBox->hide();
        ui->nextStep->show();
    }
}

void customOutput::on_nextStep_clicked()
{
    if (stepByStepAlg(lev,out,queque,flagStep)){
        ui->answer->setText(out);
        graphicConfig(BT,scene,depth);
        ui->nextStep->hide();
         ui->pushButton->show();
         ui->comboBox->show();
         ui->saveButton->show();
        return;
    }
    ui->answer->setText(out);
    graphicConfig(BT, scene,depth);
}

void customOutput::on_saveButton_clicked()
{
    QString filePath = QFileDialog::getSaveFileName(this, tr("save"), QDir::homePath(),
tr("*.txt"));
        if (QString::compare(filePath, QString()) != 0)
        {
            std::ofstream ff(qPrintable(filePath));
            ff << qPrintable(out);
            ff.close();
        }
        ui->saveButton->hide();
        close();
}
```

ПРИЛОЖЕНИЕ Е

ИСХОДНЫЙ КОД ПРОГРАММЫ. BINARYTREE.H

```
#ifndef BINARYTREE_H
#define BINARYTREE_H
#include <iostream>
#include <QVector>
#include <QList>
#include "CustomSplitForTree.h"



struct Node{
    QString info ="";
    Node* left = nullptr;
    Node* right = nullptr;
    int level = 0;
    bool flag = false;
    Node(int n = 0, Node * l = nullptr, Node * r = nullptr, int lev = 0) : info(n),
left(l), right(r),level(lev) {}
};

class BinaryTree
{
public:
    BinaryTree(){
        root = new Node;
        numbElem = 0;
    }
    Node* root = nullptr;
    Node* createTree(QStringList listTree,int& numb);
    void getNumbElem();
    int getDepth(Node* tree);
private:
    int numbElem;
};


      #endif // BINARYTREE_H
```

ПРИЛОЖЕНИЕ Ж

ИСХОДНЫЙ КОД ПРОГРАММЫ. NODEQUEQUE.CPP

```
#include "nodequeque.h"

using namespace std;

Node* BinaryTree::createTree(QStringList listTree, int& numb){
    Node* finalNode = new Node;
    if (listTree.size() == 2) return finalNode;
    QString leftTree = "",
            rightTree = "";
    int i =1;
    finalNode->info = listTree[i++];
    int indexLeftTree = i;    // Индекс открывающей скобки левого поддерева
    if (listTree[i] == "("){
        int openBrackets = 1;
        int closeBrackets = 0;
        while (openBrackets != closeBrackets){
            i++;
            if (listTree[i] == "(")
                openBrackets++;
            else if (listTree[i] == ")")
                closeBrackets++;
        }
        for (; indexLeftTree <= i; indexLeftTree++) {
            leftTree.append(listTree[indexLeftTree]);
        }
        finalNode->left = createTree(splitForTree(leftTree),numb);
        numb++;

        i++;

        if (listTree[i] == ")") { // В случае если нет правого поддерева
            return finalNode;
        }

        int indexRightTree = i;    // Индекс открывающей скобки правого поддерева
        if (listTree[i] == "("){
            int openBrackets = 1;
            int closeBrackets = 0;
            while (openBrackets != closeBrackets){
                i++;
                if (listTree[i] == "("){
                    openBrackets++;
                }
                else if (listTree[i] == ")"){
                    closeBrackets++;
                }
            }
            for (; indexRightTree <= i; indexRightTree++){
                rightTree.append(listTree[indexRightTree]);
            }
            finalNode->right = createTree(splitForTree(rightTree),numb);
            numb++;
        }
    }
    return finalNode;
```

```
}

int BinaryTree::getDepth(Node *root){
    if(root == nullptr) return 0;
    else{
        int lDepth = getDepth(root->left);
        int rDepth = getDepth(root->right);

        if (lDepth > rDepth)  return(lDepth + 1);
        else return(rDepth + 1);
    }
}
```

ПРИЛОЖЕНИЕ Е

ИСХОДНЫЙ КОД ПРОГРАММЫ. STEPBYSTEP.CPP

```
#include "stepbystep.h"

bool stepByStepAlg(int& lev, QString& out,NodeQueue* queque_,bool flagV){
    for(;;){
        Node* current = queque_->top();
        current->flag=true;
        queque_->remove();
        if (lev!=current->level){
            lev++;
            if (lev == 1)
                out+="Level:"+QString::number(lev)+'\n';
            else
                out+="\nLevel:"+QString::number(lev)+'\n';
        }
        out+=current->info+'\t';
        if (current->left==nullptr && current->right == nullptr && queque_->size()==0){
            return true;
        }
        if (current->left!=nullptr){
            queque_->addItem(current->left);
            current->left->level = current->level +1;
        }
        if (current->right!=nullptr){
            queque_->addItem(current->right);
            current->right->level = current->level +1;
        }
        if (flagV==true) return false;
    }
}
```

ПРИЛОЖЕНИЕ 3

ИСХОДНЫЙ КОД ПРОГРАММЫ. BINARYTREE.CPP

```
#include "nodequeque.h"

bool NodeQueue::isEmpty(){ //проверка очереди на пустоту
    if (last==0) return true;
    else return false;
}


void NodeQueue::addItem(Node* value){
    if (last == maxSize){
        return;
    }
     data[last++] = value;
}


Node* NodeQueue::remove(){
    Node* elem = data[0];

    for (int i =1;i<last+1;i++){
        data[i-1] = data[i];
    }
    last--;
    return elem;
}


Node* NodeQueue::top(){
    return data[0];
}

int NodeQueue::getMaxSize(){
    return maxSize;
}

int NodeQueue::size(){ //размер очереди
    return last;
}
```