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: «Деревья»

Студент гр. 8381 _____ Сосновский Д.Н.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2019

Цель работы

Ознакомиться со структурой данных «Бинарное дерево» и особенностями её реализации на языке C++. Составить программу, реализующую эту структуру данных и позволяющая пользователю работать с ней.

Постановка задачи

Для заданного бинарного дерева с произвольным типом элементов:

- вывести изображение заданного бинарного дерева;

- перечислить элементы соответствующего леса в горизонтальном порядке (в ширину).

Корнем бинарного дерева $B(F)$ является корень первого дерева T_1 в лесу F, левым поддеревом бинарного дерева $B(F)$ является бинарное дерево, представляющее лес поддеревьев первого дерева T_1, а правым поддеревом бинарного дерева $B(F)$ является бинарное дерево, представляющее лес остальных (кроме первого) деревьев исходного леса F.

Ход работы

Для выполнения данной задачи мною была разработана программа на языке C++ с использованием Qt. Для того, чтобы пользователю было проще взаимодействовать с программой, был разработан интерфейс, с помощью которого пользователь вводил исходное скобочное выражение бинарного дерева. Так же, интерфейс оснащён полем для вывода результата обхода соответствующего леса в ширину, полем для отображения рисунка бинарного дерева, а так же содержит кнопку в разделе меню «Помощь», при нажатии на которую пользователь может получить информацию о формате ввода в программу, либо о формате вывода ответа программы на запрос.

Когда пользователь ввёл исходную строку, содержащую скобочное представление бинарного дерева, ему необходимо нажать на кнопку «Перевести», чтобы получить рисунок введённого бинарного дерева и запись в ширину соответствующего леса.

Программа оснащена синтаксической проверкой корректного ввода. Таким образом, программа не поведёт себя непредсказуемо при ошибке во входных данных.

Для того, чтобы дать пользователю ответ, программа строит из скобочной последовательности бинарное дерево путём обхода строки. При этом каждый символ строки обрабатывается отдельно. Соответственно, для каждого элемента строки программа сопоставляет специальные действия. Таким образом, каждый раз встречая открывающую скобку, программа добавляет новую вершину в дерево. Далее, встречая значение, заполняет эту вершину. И, наконец, встречая закрывающую скобку, поднимается на один или несколько уровней выше (к предыдущей вершине) для того, чтобы можно было добавлять вершины и дальше.

Для того, чтобы получить обход соответствующего леса в ширину, достаточно просто пройтись по заданному бинарному дереву «скатываясь вправо». Иными словами, нужно последовательно выводить правые потомки у всех вершин, и переходить к левому потомку и продолжать вывод, если таковые кончились.

Таким образом, на выходе программа выписывает элементы соответствующего заданному бинарному дереву леса и рисует заданное бинарное дерево.

Оценка сложности работы программы

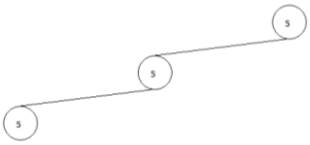
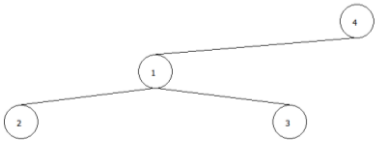
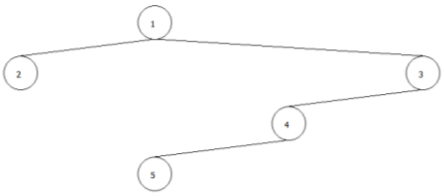
Программа имеет временную сложность $O(n)$ для построения бинарного дерева и вывода результата обхода соответствующего леса в ширину и временную сложность $O(n^2)$ для отрисовки дерева.

Вывод

В ходе выполнения данной лабораторной работы я освоил структуру данных «Бинарное дерево» и реализовал программу, позволяющую пользователю работать с ней.

ПРИЛОЖЕНИЕ

Тестирование программы

Ввод	Вывод (строка)	Вывод (рисунок)
(5(5(5)))	5, 5, 5	
(4(1(2)(3)))	4, 1, 3, 2	
(4(1(2)(3)5))	Ошибка в строке	Нет рисунка, ошибка в строке
	Пустая строка	Нет рисунка, ошибка в строке
(1(2)(3(4(5))))	1, 3, 2, 4, 5	

Графики оценки временной сложности

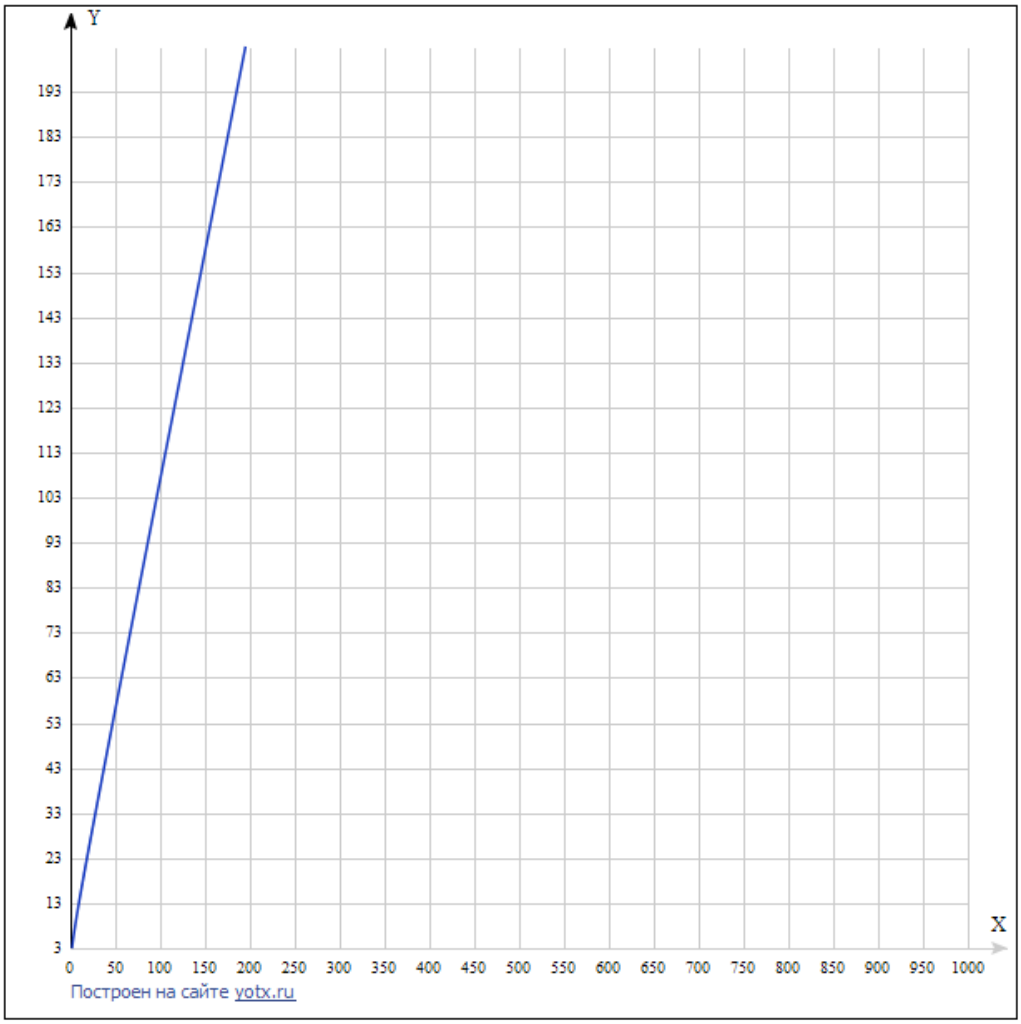


Рисунок 1 – график сложности перевода строки

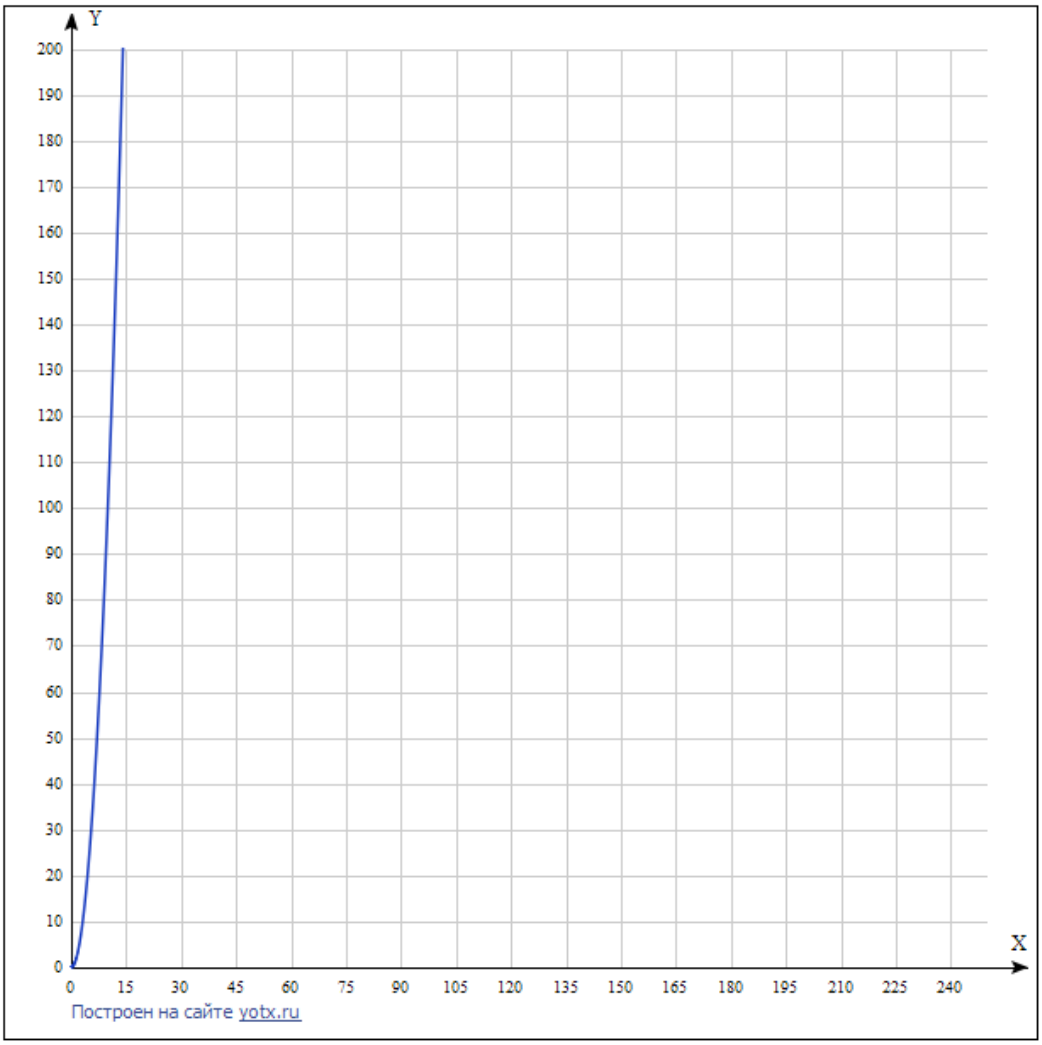


Рисунок 2 - временная сложность отрисовки бинарного дерева

Код программы

Файл binarytree.h

```
#pragma once
#include <QString>
#include <QGraphicsScene>
#include <QGraphicsTextItem>
#include "Stack.hpp"
enum symbols {LEFT_BRACKET, RIGHT_BRACKET, VALUE};

class BinaryTree
{
private:
    class Node
    {
        public:
```

```
            Node();
            ~Node();
            Node* leftNode = nullptr;
            Node* rightNode = nullptr;
            QString value = nullptr;
    };

    void constructTreeFromParenthesesString();
    symbols analyzeSymbol(int symbolPosition);
    QString getCurrentNodeValue(int& valueStartPosition);



    void drawNode(Node* node, int axis, int ordinate, int lineStartX, int
lineStartY);
    int countMaxLenLeft(Node* node);
    int countMaxLenRight(Node* node);

public:
    QGraphicsScene* scene;
    BinaryTree(QString parenthesesTreeString, QGraphicsScene* scene);
    ~BinaryTree();

    void constructNewNode(Node*& currentNode);

    Node* root = nullptr;

    QString parenthesesTreeString = nullptr;

    QString getForestBreadthString();

    void drawTree();
};
```

Файл mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QPainter>
#include <QMessageBox>
#include "binarytree.h"
namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
```

```
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);

    ~MainWindow();

private slots:
    void on_pushButton_clicked();

    void on_action_triggered();

    void on_action_2_triggered();

private:
    Ui::MainWindow *ui;
};

#endif // MAINWINDOW_H
```

Файл stack.hpp

```
#ifndef STACK_H
#define STACK_H

template<typename StackElementType>
class Stack
{
private:
    class StackElement
    {
        public:
            StackElement(StackElementType value):
                value(value)
            {

            }

            ~StackElement() { }

            StackElement* nextElement = nullptr;

            StackElementType value;
    };

    StackElement* head = nullptr;

public:

    int size = 0;
```

```
    Stack() {

    }


    ~Stack() {
        StackElement* deletableElement = head;
        while(head != nullptr)
        {
            deletableElement = head;
            head = head->nextElement;
            delete deletableElement;
        }
    }


    void add(StackElementType newElementValue) {
        if (head == nullptr)
        {
            head = new StackElement(newElementValue);
            head->nextElement = nullptr;
        } else {
            StackElement* addingStackElement = new
StackElement(newElementValue);
            addingStackElement->nextElement = head;
            head = addingStackElement;
        }
        size++;
    }


    StackElementType get() {
        StackElementType gettingElementValue = head->value;

        StackElement* nextElement = head->nextElement;
        delete head;
        head = nextElement;
        size -= 1;
        return gettingElementValue;
    }


};

#endif // STACK_H
```

Файл binarytree.cpp

```
#include "binarytree.h"
```

```
BinaryTree::BinaryTree(QString parenthesesTreeString, QGraphicsScene*
scene):
    parenthesesTreeString(parenthesesTreeString), scene(scene)
{
    if(parenthesesTreeString == nullptr) throw "Пустая строка";
    root = new Node();

    for(int i = 0; i < this->parenthesesTreeString.size(); i++)
    {
        if(this->parenthesesTreeString[i] == ' ')
        {
            this->parenthesesTreeString.remove(i, 1);
            i--;
        }
    }
    constructTreeFromParenthesesString();
}

BinaryTree::Node::Node()
{
}

BinaryTree::Node::~Node()
{
    if(leftNode != nullptr) delete leftNode;
    if(rightNode != nullptr) delete rightNode;
}

BinaryTree::~BinaryTree()
{
    delete root;
}


void BinaryTree::constructTreeFromParenthesesString()
{
    Node* currentNode = root;

    if(parenthesesTreeString[0] != '(' ||
parenthesesTreeString[parenthesesTreeString.size()-1] != ')') throw
"Ошибка в строке";
    else
    {
        parenthesesTreeString.remove(0,1);
        parenthesesTreeString.remove(parenthesesTreeString.size() - 1,
1);
    }
```

```
    Stack<Node*> stack;
    for(int i = 0; i < parenthesesTreeString.size(); i++)
    {
        symbols currentSymbol = analyzeSymbol(i);

        switch(currentSymbol)
        {
            case VALUE: {
                QString currentNodeValue = getCurrentNodeValue(i);
                if (currentNode->value != nullptr) throw "Ошибка в
строке";
                currentNode->value = currentNodeValue;
                break;
            }
            case LEFT_BRACKET: {
                stack.add(currentNode);
                constructNewNode(currentNode);
                if(currentNode->leftNode != nullptr && currentNode-
>rightNode != nullptr) throw "Ошибка в строке";
                break;
            }
            case RIGHT_BRACKET: {
                if (stack.size == 0) throw "Ошибка в строке";
                currentNode = stack.get();
                break;
            }
            default: {
                break;
            }

        }
    }

    if(currentNode != root) throw "Ошибка в строке";

    drawTree();
}

void BinaryTree::constructNewNode(Node*& currentNode)
{
    if (currentNode->leftNode == nullptr)
    {
        currentNode->leftNode = new Node();
        currentNode = currentNode->leftNode;
    }
    else if (currentNode->rightNode == nullptr)
    {
        currentNode->rightNode = new Node();
        currentNode = currentNode->rightNode;
```

```
    }
}

QString BinaryTree::getCurrentNodeValue(int & i)
{
    QString currentNodeValue = nullptr;
    while (parenthesesTreeString[i] != ')' && parenthesesTreeString[i] !=
'(' && i < parenthesesTreeString.size())
    {
            currentNodeValue.append(parenthesesTreeString[i]);
            i++;
    }
    i--;
    return currentNodeValue;
}

symbols BinaryTree::analyzeSymbol(int symbolPosition)
{
    QChar currentSymbol = parenthesesTreeString[symbolPosition];
    switch (currentSymbol.unicode()) {
        case '(': {
            return LEFT_BRACKET;
        }
        case ')': {
            return RIGHT_BRACKET;
        }
        default: {
            return VALUE;
        }
    }
}

QString BinaryTree::getForestBreadthString()
{
    bool anyNodesFound = true;
    int nodeLevelCounter = 0;
    Node* currentNode = root;
    Node* temporary = root;
    QString answerString = nullptr;

    if(root->leftNode == nullptr && root->rightNode == nullptr) return
root->value + ", ";

    while(anyNodesFound == true)
    {
        temporary = root;
        currentNode = root;
        anyNodesFound = false;
        while(currentNode != nullptr)
```

```
        {
            int i = 0;
            for(i = 0 ; i < nodeLevelCounter; i++)
            {
                if(currentNode->leftNode != nullptr)
                    currentNode = currentNode->leftNode;
                else
                {
                    break;
                }
            }

            if(i == nodeLevelCounter)
            {
                while(currentNode != nullptr)
                {
                    if(currentNode->value == nullptr) answerString +=
"#";
                    else answerString += currentNode->value;
                    answerString += ", ";
                    currentNode = currentNode->rightNode;
                }
                anyNodesFound = true;
            }

            if(nodeLevelCounter == 0) continue;

            currentNode = temporary->rightNode;
            temporary = temporary->rightNode;
        }
        nodeLevelCounter++;
    }

    return answerString;
}

void BinaryTree::drawTree()
{
    int rootAxis = 25;
    int rootOrdinate = 25;

    int leftRightMax = countMaxLenRight(root->leftNode) + 1;
    int rightLeftMax = countMaxLenLeft(root->rightNode) + 1;

    scene->addEllipse(QRectF(0, 0, 50, 50));

    QGraphicsTextItem *textItem = new QGraphicsTextItem(root->value);
    textItem->setPos(15, 15);
    scene->addItem(textItem);
```

```
    drawNode(root->leftNode, rootAxis - 100*leftRightMax, rootOrdinate +
75, rootAxis, rootOrdinate+25);
    drawNode(root->rightNode, rootAxis + 100*rightLeftMax, rootOrdinate +
75, rootAxis, rootOrdinate+25);


}

void BinaryTree::drawNode(Node *node, int axis, int ordinate, int
lineStartX, int lineStartY)
{
    if(node == nullptr) return;
    else {
        scene->addEllipse(QRectF(axis-25, ordinate-25, 50, 50));
        QGraphicsTextItem *textItem = new QGraphicsTextItem(node->value);
        textItem->setPos(axis-10, ordinate-10);
        scene->addItem(textItem);

        scene->addLine(QLine(lineStartX, lineStartY, axis, ordinate-25));

        int leftRightMax = countMaxLenRight(node->leftNode) + 1;
        int rightLeftMax = countMaxLenLeft(node->rightNode) + 1;

        drawNode(node->leftNode,  axis - 100*leftRightMax, ordinate + 75,
axis, ordinate+25);
        drawNode(node->rightNode, axis + 100*rightLeftMax, ordinate + 75,
axis, ordinate+25);
    }
}

int BinaryTree::countMaxLenLeft(Node *node)
{
   if(node == nullptr) return 0;
   else {
       int leftcnt = countMaxLenLeft(node->leftNode) + 1;
       int rightcnt = countMaxLenLeft(node->rightNode) - 1;
       if(leftcnt > rightcnt) return leftcnt;
       else return  rightcnt;
   }
}

int BinaryTree::countMaxLenRight(Node *node)
{
    if(node == nullptr) return 0;
    else {
        int leftcnt = countMaxLenRight(node->leftNode) - 1;
        int rightcnt = countMaxLenRight(node->rightNode) + 1;
        if(leftcnt > rightcnt) return leftcnt;
```

```
        else return  rightcnt;
    }
}
```

Файл main.cpp

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

Файл mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

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

void MainWindow::on_pushButton_clicked()
{
    QString inputString = ui->input->text();
    try{
        QGraphicsScene* scene = new QGraphicsScene(this);
        BinaryTree binTree(inputString, scene);
        QString answer = binTree.getForestBreadthString();
        answer.remove(answer.size()-2, 2);
        ui->output->setText(answer);
        ui->binTreePicture->setScene(binTree.scene);
    }
    catch (const char* exc)
    {
        ui->output->setText(exc);
    }
}

void MainWindow::on_action_triggered()
```

```
{
    QMessageBox::information(0, "Правила ввода", "Скобочная запись
бинарного дерева имеет вид: \n < БД > ::= < пусто > | < непустое
БД > \n < непустое БД > ::=  (< корень > < БД > < БД > ).");
}

void MainWindow::on_action_2_triggered()
{
    QMessageBox::information(0, "Формат вывода", "В поле ответа
будут выведены элементы соответствующего леса. Элементы выводятся
по ширине, то есть элементы каждого уровня выводятся слева
направо, начиная с уровня корня. \nЗнак \"#\" означает пустое
значение элемента.");
}
```