**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 |  | Нгуен Ш. Х. |
| 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, которая строит изображение леса и бинарного дерева.

**Задание**

## 6. Для заданного леса с произвольным типом элементов:

## - получить естественное представление леса бинарным деревом;

## - вывести изображение леса и бинарного дерева;

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

**Выполнение**

Написание работы производилось на базе операционной системы Windows 10 в среде разработки QtCreator с использованием фреймворка Qt. Сборка, отладка производились в QtCreator, запуск программы осуществлялся через командную строку. Исходные коды файлов программы представлены в приложениях А-М.

Для реализации программы был разработан графический интерфейс с помощью встроенного в QtCreator UI-редактора. Он представляет из себя поле ввода, кнопку считывания, поле вывода с возможностью графического отображения результата.

Структура леса :

struct Node{

bool isLeaf;

char info;

int total;

Node \*\*Tnode;

};

struct Tree{

int deep;

Node \*root;

};

struct Forest{

int count;

Tree \*\*tree;

};

## Также были реализованы функции, создающие лес из входной строки и подключены к серверной части через файл forest.cpp, использующий библиотеку forest.h, приведены в табл.1

|  |  |
| --- | --- |
| Функция | Назначение |
| Forest \*takeForest(char\* arr, int len); | Создать лес из входной строки. |
| Node \*AppendNode(Node \*node,char\* arr,int& i,int len, bool isAlpha,int total); | Создать новый узел и добавить это узел на Node\* node. |
| int CountTree(char\* arr, int len); | Подсчитывает количество деревьев, которые будут cоздать из входной строки. |
| int CheckErr(char \*arr, int len); | Возвращает значение ошибки FlagErr. |
| int BranchOfNode(char \*arr, int i, int len); | Подсчитывает количество ветвей для узла. |
| Forest \*createForest(int count); | Создать лес с count деревьев. |
| Tree \*createTree(); | Создать новое дерево. |
| Node \*createNode(char info,bool isAlpha, int total); | Создать новый узел |

|  |  |
| --- | --- |
| int DeepOfTree(char \*arr, int i , int len); | Возвращает глубину дерева |
| void takeInfoOfNode(Node \*root, string &out); | Получить информацию о дереве через корень root дерева и сохраняются в string out. |

Таблица 1– Основные функции создания лес из входной строки

Далее получим естественное представление леса бинарным деревом через функции, приведены в табл.2 с структурой бинарного дерева :

struct BinNode

{

bool isLeaf;

char info;

BinNode \*left;

BinNode \*right;

};

struct BinTree

{

BinNode \*root;

int deep;

};

|  |  |
| --- | --- |
| Функция | Назначение |
| BinTree \*createBinTree(); | Создать новое бинарное дерево. |
| BinNode \*createBinNode(char info, bool isAlpha); | Создать новый бинарный узел. |
| BinTree \*createBTFromForest(Forest\* forest); | Создать бинарное дерево из леса. |
| BinNode \*ConsBT(BinNode \*root, Forest \*left, Forest \*right); | Создать полный бинарный узел. |
| Tree \*Head(Forest \*forest); | Возвращает первое дерево леса. |
| BinNode \*Root(Tree\* tree); | Возвращает корень root любого дерева. |
| Forest \*Listing(Tree \*tree); | Создать лес деревьев из корневых узлов корня root. |
| Tree \*NodeToTree(Node \*node); | Превращает структуру узла в дерево. |
| int CountDeepOfTree(Node\* root); | Возвращает глубину дерева. |
| void takeInfoBT(BinNode \*root, string &out); | Возвращает информацию о дереве и сохраняется в string out. |
| void takeInfoByDeep(Node \*root,int deep,string &out); | Возвращает информацию о дереве и хранится в string out в зависимости от глубины дерева deep. |

табл.2 - Основные функции создания бинарного дерева

**Оценка эффективности алгоритма**

Алгоритм создания бинарного дерева по строке является итеративным, каждый элемент строки обрабатывается один раз, а значит сложность алгоритма можно оценить как O(N).

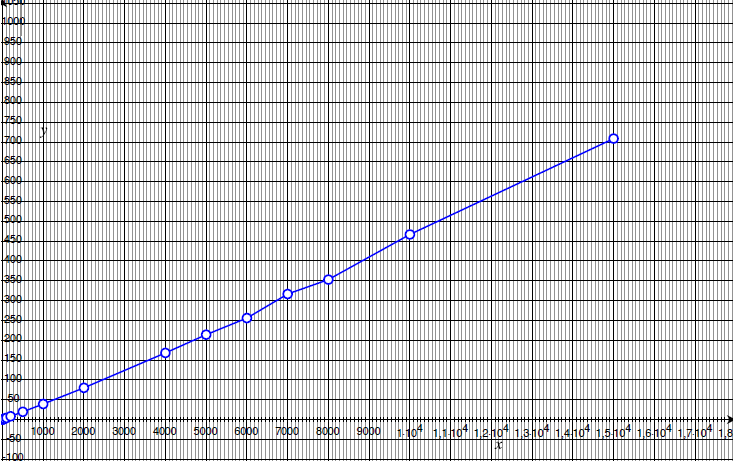


График 1 — Зависимость количества элементов к времени строительства (sec\**1e+5*);

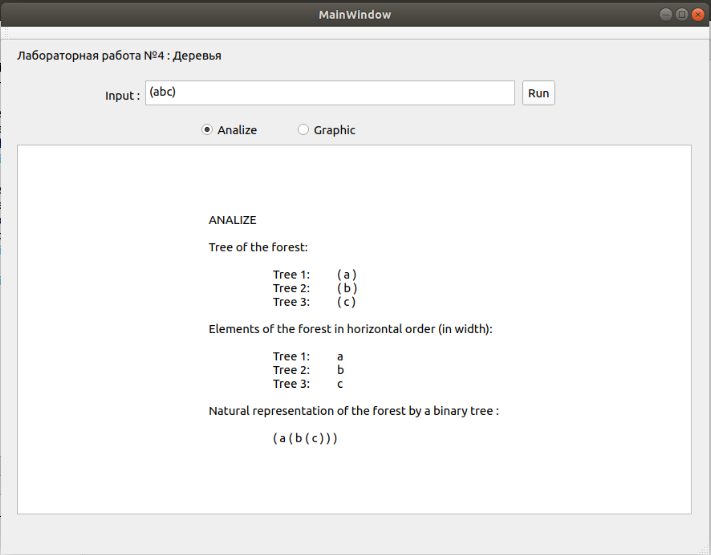
Алгоритм вывода элементов дерева в горизонтальном порядке является рекурсивным, каждый узел дерева обрабатывается один раз, следовательно, сложность алгоритма также .

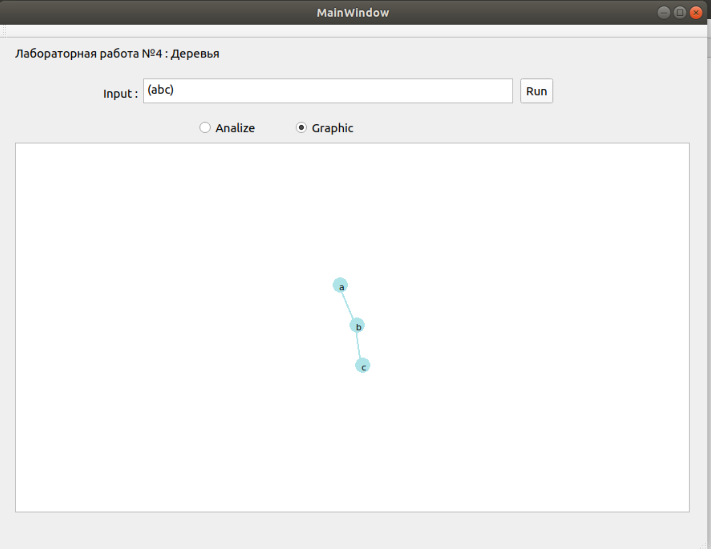
**Выводы**

В ходе выполнения лабораторной работы была написана программа, создающая естественное представление леса бинарным деревом, таже изображение бинарного дерева и перечислить элементы леса в горизонтальном порядке (в ширину).

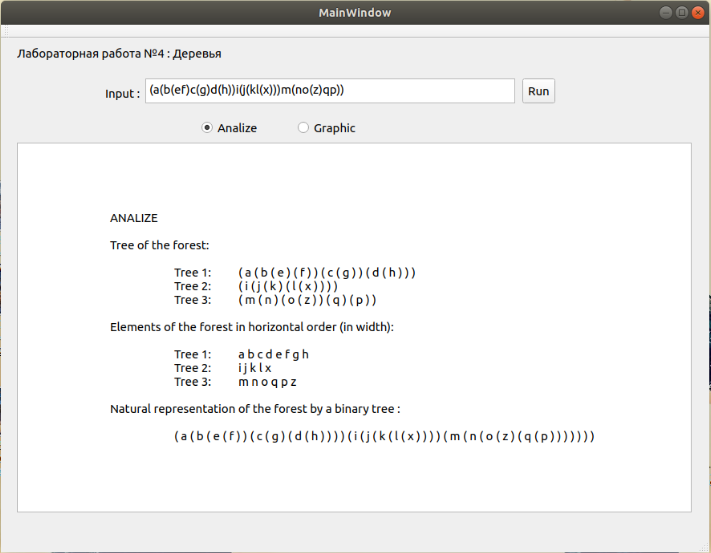
**Тестирование программы**

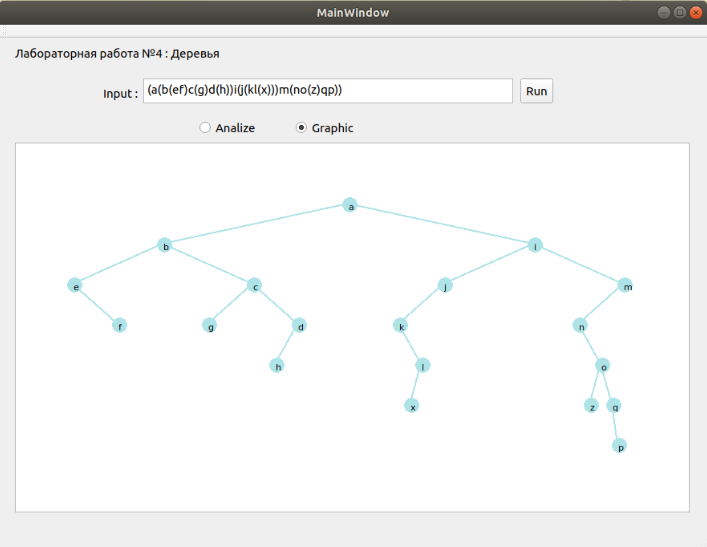
1. Лес : (abc)



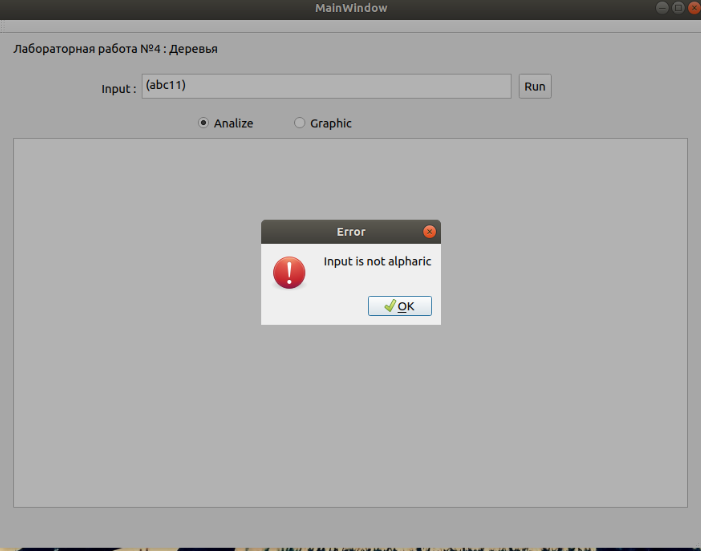


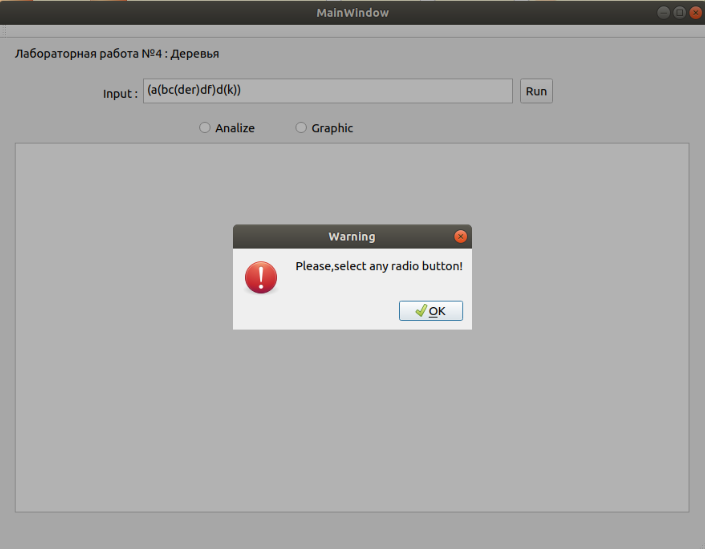
1. Лес : (a(b(ef)c(g)d(h))i(j(kl(x)))m(no(z)qp))





1. Ошибки





**Приложение А**

**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.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <console.h>

#include <QMessageBox>

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = nullptr);

~MainWindow();

console \*Console;

bool flagCase\_1 = false;

bool flagCase\_2 = false;

int flagErr;

private slots:

void on\_pushButton\_clicked();

void on\_radioButton\_2\_clicked(bool checked);

void on\_radioButton\_clicked(bool checked);

private:

Ui::MainWindow \*ui;

QGraphicsView \*scene;

};

#endif // MAINWINDOW\_H

**Файл Mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

Console = new console;

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::on\_pushButton\_clicked()

{

QString data = ui->textEdit->toPlainText();

QGraphicsScene \*scene = new QGraphicsScene(ui->graphicsView);

scene->clear();

flagErr = Console->checkErr(data);

if(flagErr == 1){

QMessageBox::warning(this,"Err","Missing character '(' or ')'");

}

else if(flagErr == 2){

QMessageBox::warning(this,"Err","Tree empty.");

}

else if(flagErr == 3){

QMessageBox::warning(this,"Error","Input is not alpharic");

}

else if(flagErr == 4){

QMessageBox::warning(this,"Error","Extra character ')'");

}

else if (flagErr == 5){

QMessageBox::warning(this,"Error","The first tree in forest incorrect.");

}

else if (flagErr == 6) {

QMessageBox::warning(this,"Error","Input incorrect");

}

else if (flagErr == 0) {

if(flagCase\_1 == false && flagCase\_2 == false)

QMessageBox::warning(this,"Warning","Please,select any radio button!");

else if(flagCase\_2 == true){

scene = Console->Console( data);

ui->graphicsView->setScene(scene);

}

else if(flagCase\_1 == true){

scene = Console->Analize(data);

ui->graphicsView->setScene(scene);

}

}

}

void MainWindow::on\_radioButton\_2\_clicked(bool checked)

{

flagCase\_2 = checked;

flagCase\_1 = false;

}

void MainWindow::on\_radioButton\_clicked(bool checked)

{

flagCase\_2 = false;

flagCase\_1 = checked;

}

**Файл forest.h**

#include <iostream>

#include <string>

#include <ctype.h>

#include <fstream>

#include <cstring>

using namespace std;

struct Node{

bool isLeaf;

char info;

int total;

Node \*\*Tnode;

};

struct Tree{

int deep;

Node \*root;

};

struct Forest{

int count;

Tree \*\*tree;

};

Forest \*takeForest(char\* arr, int len);

Node \*AppendNode(Node \*node,char\* arr,int& i,int len, bool isAlpha,int total);

int CountTree(char\* arr, int len);

int CheckErr(char \*arr, int len);

int BranchOfNode(char \*arr, int i, int len);

Forest \*createForest(int count);

Tree \*createTree();

Node \*createNode(char info,bool isAlpha, int total);

int DeepOfTree(char \*arr, int i , int len);

void takeInfoOfNode(Node \*root, string &out);

**Файл forest.cpp**

#include "forest.h"

// Viết 1 hàm loại bỏ dấu cách trong dữ liệu đầu vào

Forest \*takeForest(char\* arr, int len){

int countTree = CountTree(arr,len);

Forest \*forest = createForest(countTree);// Khoi tao countTree Tree

int i = 1;

for(int j = 0; j < forest->count; j++){

//Crete root of tree

int deep = DeepOfTree(arr,i,len);

int total = BranchOfNode(arr,i,len);

Node \*root = createNode(arr[i],true,total);

if(total>0){

i+=2;

}

AppendNode(root,arr,i,len,true,total);

forest->tree[j]->root = root;

forest->tree[j]->deep = deep;

}

return forest;

}

Node \*AppendNode(Node \*node,char\* arr,int& i,int len, bool isAlpha,int total){// i = vi tri cua root

/\* ket thuc ham nay , gia tri cua i = vi tri cua root of second tree\*/

if(total == 0){

if(isalpha(arr[i+1]))

i++;

else {

i++;

while(arr[i] == ')'){

i++;

}

}

return node;

}

for(int k = 0; k < total; k++){

Node \*temp = new Node;

int countNode = BranchOfNode(arr,i,len);

temp = createNode(arr[i],isAlpha,countNode);

if(countNode > 0){

i+=2;

}

AppendNode(temp,arr,i,len,isAlpha,countNode);

node->Tnode[k] = temp;

}

return node;

}

int DeepOfTree(char \*arr, int i , int len){// i = vi tri cua root cua tree

if(arr[i+1]!='(')

return 0;

int delta = 0;

int deep = delta;

int flag = i;

for(int j = i; j<len; j++){

if(arr[j] == '('){

delta += 1;

deep = delta>deep?delta:deep;

}

else if(arr[j] == ')')

delta -= 1;

if(delta == 0 && j != flag)

return deep+1;

}

}

/\* function checked \*/

int CountTree(char\* arr, int len){

int countTree = 0;

int Delta = 0; //hieu cua ( va )

for(int i = 0; i < len; i++){

if(arr[i] == '(')

Delta += 1;

else if(arr[i] == ')')

Delta -= 1;

if(Delta == 1 && arr[i] != ')')

countTree += 1;

}

return countTree-1;

}

int CheckErr(char \*arr, int len){

for(int i = 0; i < len; i++){

if(isalpha(arr[i]) == false && arr[i] != ')' && arr[i] != '(')

return 3; // 3- Input incorrect

}

if(!isalpha(arr[1]))

return 5;

int count = 0;

for(int i = 0; i < len; i++){

if(count < 0)

return 4; // thua dau ')'

if(arr[i] == '(')

count++;

else if(arr[i] == ')')

count--;

if(count == 0 && i <len -1)

return 6;

}

if(count != 0)

return 1; // 1- thieu ( hoac )

for(int i = 0; i < len; i++){

if(arr[i] =='(' && arr[i+1] == ')')

return 2; // 2- tree empty

}

return 0;

}

/\*Function checked\*/

int BranchOfNode(char \*arr, int i, int len){// i = vi tri cua Root cua Tree

int count = 0;

int flag = i;

int delta = 0;

if(arr[i+1] != '(') // TH ab => branch = 0

return 0;

else{

for(int j = i+1; j < len ; j++){

if(arr[j] == '(')

delta += 1;

else if(arr[j] == ')'){

delta -= 1;

if(delta == 0 && j != flag)

return count;

}

else if(isalpha(arr[j]) && delta == 1){

count++;

}

}

}

}

Forest \*createForest(int count){

Forest \*forest = new Forest;

forest->count = count;

forest->tree = new (Tree\*);

for(int i = 0; i<count; i++){

forest->tree[i] = new Tree;

forest->tree[i] = createTree();

}

return forest;

}

Tree \*createTree(){

Tree \*tree = new Tree;

tree->root = nullptr;

tree->deep = 0;

return tree;

}

Node \*createNode(char info,bool isAlpha, int total){

Node \*node = new Node;

node->info = info;

node->isLeaf = isAlpha;

node->total = total;

node->Tnode = new (Node\*);

for(int j = 0; j < total; j++){

node->Tnode[j] = new Node;

node->Tnode[j] = nullptr;

}

return node;

}

void takeInfoOfNode(Node \*root, string &out){

out += "( ";

out.append(1,root->info);

out+=" ";

for(int i = 0; i < root->total; i++){

takeInfoOfNode(root->Tnode[i],out);

}

out += ") ";

}

**Файл bintree.h**

#include "forest.h"

struct BinNode

{

bool isLeaf;

char info;

BinNode \*left;

BinNode \*right;

};

struct BinTree

{

BinNode \*root;

int deep;

};

BinTree \*createBinTree();

BinNode \*createBinNode(char info, bool isAlpha);

BinTree \*createBTFromForest(Forest\* forest);

BinNode \*ConsBT(BinNode \*root, Forest \*left, Forest \*right);

Tree \*Head(Forest \*forest);

Forest \*Tail(Forest \*forest);

BinNode \*Root(Tree\* tree);

Forest \*Listing(Tree \*tree);

Tree \*NodeToTree(Node \*node);

int CountDeep(BinNode \*&node);

int CountDeepOfTree(Node\* root);

void takeInfoBT(BinNode \*root, string &out);

void takeInfoByDeep(Node \*root,int deep,string &out);

**Файл bintree.cpp**

#include "bintree.h"

BinTree \*createBinTree()

{

BinTree \*tree = new BinTree;

tree->root = nullptr;

tree->deep = 0;

return tree;

}

BinNode \*createBinNode(char info, bool isAlpha)

{

BinNode \*node = new BinNode;

node->info = info;

node->isLeaf = isAlpha;

node->left = nullptr;

node->right = nullptr;

return node;

}

BinTree \*createBTFromForest(Forest\* forest){

BinTree \*bintree = createBinTree();

bintree->root = ConsBT(Root(Head(forest)),Listing(Head(forest)),Tail(forest));

bintree->deep = CountDeep(bintree->root) - 1;

return bintree;

}

/\*checked\*/

Tree \*Head(Forest \*forest){

Tree \*tree = new Tree;

tree = forest->tree[0];// da bao gom deep

return tree;

}

/\*checked\*/

Forest \*Tail(Forest \*forest){

if(forest->count == 1)

return nullptr;

Forest \*temp = createForest((forest->count)-1); // đã khởi tạo forest cùng forest->count

for(int i = 0; i < (forest->count)-1; i++){

temp->tree[i] = forest->tree[i+1];

}

return temp;

}

/\* \*/

BinNode \*Root(Tree\* tree){// create Binnode from tree;

if(tree == nullptr)

return nullptr;

BinNode \*root = createBinNode(tree->root->info,tree->root->isLeaf);

return root;

}

Forest \*Listing(Tree \*tree){

if(tree->deep == 0) //

return nullptr;

Forest \*forest = createForest((tree->root->total));

for(int i = 0; i < forest->count; i++){

forest->tree[i] = NodeToTree(tree->root->Tnode[i]);//da tao deepoftree

}

return forest;

}

Tree \*NodeToTree(Node \*node){

Tree \*tree = createTree();

tree->root = node;

tree->deep = CountDeepOfTree(node) - 1;

return tree;

}

BinNode \*ConsBT(BinNode \*root, Forest \*left, Forest \*right){//tạo binnote\* root từ treenote thong qua ham createbinnode;

BinNode \*binnode = root;

if(left == nullptr)

binnode->left = nullptr;

else

binnode->left = ConsBT(Root(Head(left)),Listing(Head(left)),Tail(left));

if(right == nullptr)

binnode->right = nullptr;

else

binnode->right = ConsBT(Root(Head(right)),Listing(Head(right)),Tail(right));

return binnode;

}

int CountDeep(BinNode \*&node)

{

if (node == nullptr)

return 0;

int cl = CountDeep(node->left);

int cr = CountDeep(node->right);

return 1 + ((cl>cr)?cl:cr);

}

/\*checked\*/

int CountDeepOfTree(Node\* root){

if(root == nullptr)

return 0;

int max = 0;

for(int i = 0; i < root->total; i++){

int count = CountDeepOfTree(root->Tnode[i]);

max = count>max?count:max;

}

return max+1;

}

void takeInfoBT(BinNode \*root, string &out){

if(root == nullptr)

return;

out += "( ";

out.append(1,root->info);

out+=" ";

takeInfoBT(root->left,out);

takeInfoBT(root->right,out);

out += ") ";

}

/\*

void takeInfoByDeep(BinNode \*root,string &out, int deep){

if(deep == 0){

out.append(1,root->info);

out += " ";

}

else {

takeInfoByDeep(root->left,out,deep-1);

takeInfoByDeep(root->right,out,deep-1);

}

}\*/

void takeInfoByDeep(Node \*root,int deep,string &out){// dieu kien la deep <= DeepOfTree

if(deep == 0){

out.append(1,root->info);

out+= " ";

return;

}

if(root->total == 0)

return;

for(int i = 0; i<root->total; i++){

takeInfoByDeep(root->Tnode[i],deep-1,out);

}

return;

}

**Файл console.h**

#ifndef CONSOLE\_H

#define CONSOLE\_H

#include <bintree.h>

#include <QGraphicsScene>

#include <QGraphicsView>

#include <cmath>

#include <QGraphicsTextItem>

#include <QString>

#include <QMessageBox>

#include <QLineEdit>

class console

{

public:

console();

QGraphicsScene \*Console(QString data);

QGraphicsScene \*Analize(QString data);

int checkErr(QString data);

void treePainter(QGraphicsScene \*&scene, BinNode \*binnode, int w, int h, int wDelta, int hDelta, QPen &pen, QBrush &brush, QFont &font, int depth);

};

#endif // CONSOLE\_H

**Файл console.cpp**

#include "console.h"

console::console()

{

}

int console::checkErr(QString data){

int flagErr = 0;

string input = data.toStdString(); // Qstring to string

char \*arr = new char[input.length()+1];// khoi tao char\* arr

std::strcpy(arr,input.c\_str()); // string to char\*

int len = static\_cast<int>(input.length());

flagErr = CheckErr(arr,len); // check lai ham nay

return flagErr;

}

QGraphicsScene\* console::Analize(QString data){

QGraphicsScene \*scene = new QGraphicsScene;

string input = data.toStdString(); // Qstring to string

char \*arr = new char[input.length()+1];// khoi tao char\* arr

std::strcpy(arr,input.c\_str()); // string to char\*

int len = static\_cast<int>(input.length());

Forest \*forest = takeForest(arr,len);

BinTree \*bintree = createBTFromForest(forest);

string out;

out += "ANALIZE";

out += "\n\nTree of the forest:\n";

for(int i = 0; i <forest->count; i++){

out += "\n\tTree " + std::to\_string(i+1)+":\t";

takeInfoOfNode(forest->tree[i]->root,out);

}

out += "\n\nElements of the forest in horizontal order (in width):\n\t";

for(int i = 0; i <forest->count; i++){

out += "\n\tTree " + std::to\_string(i+1)+":\t";

for(int j = 0; j <= forest->tree[i]->deep; j++){

takeInfoByDeep(forest->tree[i]->root,j,out);

}

}

out+= "\n\nNatural representation of the forest by a binary tree :\n\n\t";

takeInfoBT(bintree->root,out);

scene->addText(QString::fromStdString(out));

return scene;

}

QGraphicsScene\* console::Console(QString data){

QGraphicsScene \*scene = new QGraphicsScene;

string input = data.toStdString(); // Qstring to string

char \*arr = new char[input.length()+1];// khoi tao char\* arr

std::strcpy(arr,input.c\_str()); // string to char\*

int len = static\_cast<int>(input.length());

Forest \*forest = takeForest(arr,len);

BinTree \*bintree = createBTFromForest(forest);

scene->clear();

QPen pen;

QColor color;

color.setRgb(174, 227, 232);

pen.setColor(color);

QBrush brush (color);

QFont font("Helvetica [Cronyx]", 8, 10,false);

pen.setWidth(2);

int wDeep = static\_cast<int>(pow(2, bintree->deep)+2);

int hDelta = 50;

int wDelta = 7;

int width = (wDelta\*wDeep)/2;

treePainter(scene, bintree->root, width/2, hDelta, wDelta, hDelta, pen, brush, font, wDeep);

return scene;

}

void console::treePainter(QGraphicsScene \*&scene, BinNode \*binnode, int w, int h, int wDelta, int hDelta, QPen &pen, QBrush &brush, QFont &font, int depth)

{

if (binnode == nullptr)

return ;

QString out;

out += binnode->info;

QGraphicsTextItem \*textItem = new QGraphicsTextItem;

textItem->setPos(w, h);// set toa do (x;y) cua nut

textItem->setPlainText(out);

textItem->setFont(font);

scene->addEllipse(w-wDelta/2, h, wDelta\*5/2, wDelta\*5/2, pen, brush); // Tạo hình tròn của các nút

if (binnode->left != nullptr)

scene->addLine(w+wDelta/2, h+wDelta, w-(depth/2)\*wDelta+wDelta/2, h+hDelta+wDelta, pen);

if (binnode->right != nullptr)

scene->addLine(w+wDelta/2, h+wDelta, w+(depth/2)\*wDelta+wDelta/2, h+hDelta+wDelta, pen);

scene->addItem(textItem);

treePainter(scene, binnode->left, w-(depth/2)\*wDelta, h+hDelta, wDelta, hDelta, pen, brush, font, depth/2);

treePainter(scene, binnode->right, w+(depth/2)\*wDelta, h+hDelta, wDelta, hDelta, pen, brush, font, depth/2);

return ;

}