Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе №20**

Дисциплина: «Основы алгоритмизации и программирования»

Тема: “ Бинарные деревья.”

Выполнил:

Студент группы ИВТ-20-2Б Брейкин Алексей

Проверила:

Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2021

**Постановка задачи**

1. Сформировать идеально сбалансированное бинарное дерево.
2. Тип информационного поля char. Найти количество таких элементов в дереве
3. Распечатать полученное дерево.
4. Выполнить обработку дерева в соответствии с заданием, вывести полученный результат.
5. Преобразовать идеально сбалансированное дерево в дерево поиска.  
   Распечатать полученное дерево.

**Анализ задачи**

1. Для решения задач необходимо:
   1. Организовать класс Node с полями под указатель на правый, левый и родительский узлы. А так же координаты этого узла и уровень на котором находится узел
   2. Подключить библиотеку #include "GL/freeglut.h" для печати дерева OpenGL.
   3. Организовать класс Tree, который будет хранить само дерево состоящее из нод
2. В ходе работы были использованы следующие типы данных:
   1. Класс Node c полями которые указывают на левый, правый, а так же родительский узел. Информационное поле типа char. Координаты ноды, а также уровень на котором хранится нода

#pragma once

class Node

{

public:

char value;

Node\* parent = nullptr;

Node\* left = nullptr;

Node\* right = nullptr;

Node(char value);

Node(Node\* parent, char value);

~Node();

int level = 1;

int radius = 80;

double x;

double y;

void draw();

};

* 1. Класс Tree, который хранит само дерево непосредственно.

class Tree

{

private:

bool isSearch = false;

void getArrayRecurcive(Node\*, vector<char>& res);

Node\* getLast(Node\*, int, int);

Node\* findRecurcive(Node\* node, char value);

public:

Tree();

Node\* root = nullptr;

int nodeRadius = 1;

int levelHeight = 0;

virtual void insertBalanced(char value);

void update();

void update(Node\* node, int col);

void draw();

void draw(Node\* node);

int getHeight();

int getHeight(Node\* node);

vector<char> getArray();

void reorder(Node\* node, char tmp);

};

* 1. Int
  2. String
  3. Char
  4. Указатель на ноду Node\* parent = nullptr;
  5. Вектор vector<char> arr = { 'q', 'w', 'e', 'r', 't', 'y', 'u'};

1. Для решения задачи данные были представлены в виде:
   1. Бинарное дерево создается с помощью метода insertBalanced

void Tree::insertBalanced(char value) {

if (root == nullptr)

{

root = new Node(value);

}

else {

int h = getHeight();

Node\* cur = getLast(root, 1, h);

if (cur == nullptr)

{

cur = root;

while (cur->left != nullptr)

{

cur = cur->left;

}

}

if (cur->left == nullptr) {

cur->left = new Node(cur, value);

}

else if (cur->right == nullptr) {

cur->right = new Node(cur, value);

}

}

update();

}

1. Для операций ввода и вывода использовались следующие операторы и функции:
   1. Для вывода дерева в отдельное окно используется метод draw() класса Tree. Используются методы, прописанные в библиотеке glut.h, такие как glutInit() для инициализации glut; glutInitWindowPosition() для указания позиции окна, glutWindowSize() для указания размеров окна в пикселях, glutDisplayMode() для включения двойной буферизации и для отображения графической информации через 3 основных цвета; glutCreateWidnow() для создания самого окна с названием, отправляемым в качестве параметра; glutReshapeFunc() для задания функции обработки изменения размеров окна, в качестве параметра отправляется указатель на функцию; glutDisplayFunc() задаёт функцию рисования изображения; glutMainLoop() – запуск главного цикла.

void Tree::draw(Node\* node) {

if (node == nullptr) return;

node->draw();

draw(node->left);

draw(node->right);

}

inline void drawLine(double x1, double y1, double x2, double y2, Color color) {

glColor3f(color.red, color.green, color.blue);

glBegin(GL\_LINES);

glVertex2f((GLfloat)x1, (GLfloat)y1);

glVertex2f(x2, y2);

glEnd();

}

inline void drawCircle(double x, double y, int radius, Color main, Color stroke)

{

int iterations = 1000;

glColor3f(main.red, main.green, main.blue);

glBegin(GL\_POLYGON);

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

double x1 = radius \* cos(2 \* PI \* (i / (iterations \* 1.0))) + x;

double y1 = radius \* sin(2 \* PI \* (i / (iterations \* 1.0))) + y;

glVertex2f(x1, y1);

}

glEnd();

glColor3f(stroke.red, stroke.green, stroke.blue);

glBegin(GL\_LINE\_LOOP);

for (int i = 0; i < 1000; i++) {

double x1 = radius \* cos(2 \* PI \* (i / (iterations \* 1.0))) + x;

double y1 = radius \* sin(2 \* PI \* (i / (iterations \* 1.0))) + y;

glVertex2f(x1, y1);

}

glEnd();

}

void Node::draw()

{

glLineWidth(1);

if (left) {

drawLine(x, y, left->x, left->y, lineColor);

}

if (right) {

drawLine(x, y, right->x, right->y, lineColor);

}

glLineWidth(1);

drawCircle(x, y, radius, circleColor, strokeColor);

glColor3f(0, 0, 0);

glRasterPos2f(x - glutBitmapWidth(GLUT\_BITMAP\_TIMES\_ROMAN\_24, value) / 2, y);

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24, value);

}

1. Поставленные задачи будут решены следующими действиями:
   1. Нахождение количества элементов равных ключу происзводится

for (const auto &x : arr)

{

if (tmp == x)

{

count\_1++;

}

tree.insertBalanced(x);

count\_2++;

}

* 1. Глубина дерева вычисляется:

cout << "Количество искомых символов в дереве: " << count\_1 << "\n\n\n";

count\_2 = count\_2 / 2;

cout << "Глубина дерева: " << count\_3 << "\n\n\n";

count\_1 = 0;

* 1. В функции Tree::insertBalanced(char value) производится добавление нод в дерево.

void Tree::insertBalanced(char value) {

if (root == nullptr)

{

root = new Node(value);

}

else {

int h = getHeight();

Node\* cur = getLast(root, 1, h);

if (cur == nullptr)

{

cur = root;

while (cur->left != nullptr)

{

cur = cur->left;

}

}

if (cur->left == nullptr) {

cur->left = new Node(cur, value);

}

else if (cur->right == nullptr) {

cur->right = new Node(cur, value);

}

}

update();

}

* 1. Функция которая вычисляет текущую глубину

int Tree::getHeight() {

return getHeight(root);

}

int Tree::getHeight(Node\* node) {

if (node == nullptr) return 0;

return fmax(getHeight(node->left) + 1, getHeight(node->right) + 1);

}

**Блок-схема**

















**Код**

**Source.cpp**

#include <iostream>

#include "GL/freeglut.h"

#include <string>

#include <vector>

#include "Tree.h"

using namespace std;

extern int WINDOW\_WIDTH = 1000;

extern int WINDOW\_HEIGHT = 600;

Tree tree;

void initGL() {

glLoadIdentity();

glShadeModel(GL\_SMOOTH);

glEnable(GL\_LINE\_SMOOTH);

glMatrixMode(GL\_PROJECTION);

glClearColor(1, 1, 1, 1);

glViewport(0, 0, WINDOW\_WIDTH, WINDOW\_HEIGHT);

gluOrtho2D(0, WINDOW\_WIDTH, 0, WINDOW\_HEIGHT);

}

void onReshape(int w, int h)

{

WINDOW\_WIDTH = w;

WINDOW\_HEIGHT = h;

glViewport(0, 0, WINDOW\_WIDTH, WINDOW\_HEIGHT);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, WINDOW\_WIDTH, 0, WINDOW\_HEIGHT);

tree.update();

glutPostRedisplay();

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

tree.draw();

glutSwapBuffers();

}

int count\_1 = 0;

int count\_2 = 0;

int count\_3 = 1;

int main(int argc, char\* argv[])

{

setlocale(LC\_ALL, "Ru");

vector<char> arr = { 'q', 'w', 'e', 'r', 't', 'y', 'u'};

char tmp;

cout << "Искомый символ: ";

cin >> tmp;

for (const auto &x : arr)

{

if (tmp == x)

{

count\_1++;

}

tree.insertBalanced(x);

count\_2++;

}

while (count\_2 != 1)

{

count\_2 = count\_2 / 2;

count\_3++;

}

cout << "Количество искомых символов в дереве: " << count\_1 << "\n\n\n";

count\_2 = count\_2 / 2;

cout << "Глубина дерева: " << count\_3 << "\n\n\n";

count\_1 = 0;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA);

glutInitWindowSize(WINDOW\_WIDTH, WINDOW\_HEIGHT);

glutCreateWindow("Бинарное дерево");

initGL();

glutReshapeFunc(onReshape);

glutDisplayFunc(display);

glutMainLoop();

return 0;

}

**Node.cpp**

#include "Node.h"

#include "GL/freeglut.h"

#include "math.h"

#include <string.h>

#define PI 3.14159265358979323846

struct Color {

double red;

double green;

double blue;

};

static Color lineColor{ 0, 0, 0 };

static Color circleColor{ 1, 1, 1 };

static Color defTextColor{ 0, 0, 0 };

static Color strokeColor{ 0, 0, 0 };

Node::Node(char value) {

this->value = value;

}

Node::Node(Node\* parent, char value) {

this->parent = parent;

this->value = value;

}

Node::~Node()

{

}

inline void drawLine(double x1, double y1, double x2, double y2, Color color) {

glColor3f(color.red, color.green, color.blue);

glBegin(GL\_LINES);

glVertex2f((GLfloat)x1, (GLfloat)y1);

glVertex2f(x2, y2);

glEnd();

}

inline void drawCircle(double x, double y, int radius, Color main, Color stroke)

{

int iterations = 1000;

glColor3f(main.red, main.green, main.blue);

glBegin(GL\_POLYGON);

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

double x1 = radius \* cos(2 \* PI \* (i / (iterations \* 1.0))) + x;

double y1 = radius \* sin(2 \* PI \* (i / (iterations \* 1.0))) + y;

glVertex2f(x1, y1);

}

glEnd();

glColor3f(stroke.red, stroke.green, stroke.blue);

glBegin(GL\_LINE\_LOOP);

for (int i = 0; i < 1000; i++) {

double x1 = radius \* cos(2 \* PI \* (i / (iterations \* 1.0))) + x;

double y1 = radius \* sin(2 \* PI \* (i / (iterations \* 1.0))) + y;

glVertex2f(x1, y1);

}

glEnd();

}

void Node::draw()

{

glLineWidth(1);

if (left) {

drawLine(x, y, left->x, left->y, lineColor);

}

if (right) {

drawLine(x, y, right->x, right->y, lineColor);

}

glLineWidth(1);

drawCircle(x, y, radius, circleColor, strokeColor);

glColor3f(0, 0, 0);

glRasterPos2f(x - glutBitmapWidth(GLUT\_BITMAP\_TIMES\_ROMAN\_24, value) / 2, y);

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24, value);

}

**Tree.cpp**

#include "Tree.h"

#include "Tree.h"

#include "Node.h"

#include <math.h>

#include <algorithm>

#include <iostream>

extern int WINDOW\_WIDTH;

extern int WINDOW\_HEIGHT;

const int MIN\_RADIUS = 10;

const int MAX\_RADIUS = 30;

Tree::Tree(void)

{

}

void Tree::insertBalanced(char value) {

if (root == nullptr)

{

root = new Node(value);

}

else {

int h = getHeight();

Node\* cur = getLast(root, 1, h);

if (cur == nullptr)

{

cur = root;

while (cur->left != nullptr)

{

cur = cur->left;

}

}

if (cur->left == nullptr) {

cur->left = new Node(cur, value);

}

else if (cur->right == nullptr) {

cur->right = new Node(cur, value);

}

}

update();

}

Node\* Tree::getLast(Node\* node, int level, int lastLevel) {

if (level == lastLevel)

return nullptr;

if (level < lastLevel && node->left == nullptr || node->right == nullptr)

return node;

Node\* tmp = getLast(node->left, level + 1, lastLevel);

if (tmp != nullptr)

return tmp;

return getLast(node->right, level + 1, lastLevel);

}

int Tree::getHeight() {

return getHeight(root);

}

int Tree::getHeight(Node\* node) {

if (node == nullptr) return 0;

return fmax(getHeight(node->left) + 1, getHeight(node->right) + 1);

}

void Tree::update() {

// update root node

int levels = getHeight();

if (levels == 0)

{

return;

}

levelHeight = WINDOW\_HEIGHT / getHeight();

nodeRadius = fmin(fmin(((WINDOW\_WIDTH / pow(2, levels) \* 1.0) / 2) \* 0.8, ((WINDOW\_HEIGHT / levels) / 2) \* 0.8), MAX\_RADIUS);

nodeRadius = fmax(nodeRadius, MIN\_RADIUS);

update(root, 1);

}

void Tree::update(Node\* node, int col) {

if (node == nullptr) return;

if (node->parent == nullptr) {

node->level = 1;

}

else {

node->level = node->parent->level + 1;

}

node->radius = nodeRadius;

int absCol = col - pow(2, node->level - 1) + 1;

double ww = ((WINDOW\_WIDTH) / pow(2, node->level - 1));

node->x = ww \* (absCol - 1) + ww / 2;

node->y = WINDOW\_HEIGHT - (node->level \* levelHeight - levelHeight / 2);

update(node->left, col << 1);

update(node->right, (col << 1) | 1);

}

void Tree::draw(Node\* node) {

if (node == nullptr) return;

node->draw();

draw(node->left);

draw(node->right);

}

void Tree::draw() {

draw(root);

}

**Node.h**

#pragma once

class Node

{

public:

char value;

Node\* parent = nullptr;

Node\* left = nullptr;

Node\* right = nullptr;

Node(char value);

Node(Node\* parent, char value);

~Node();

int level = 1;

int radius = 80;

double x;

double y;

void draw();

};

**Tree.h**

#pragma once

#include <iostream>

#include "Node.h"

#include <vector>

using namespace std;

class Tree

{

private:

bool isSearch = false;

void getArrayRecurcive(Node\*, vector<char>& res);

Node\* getLast(Node\*, int, int);

Node\* findRecurcive(Node\* node, char value);

public:

Tree();

Node\* root = nullptr;

int nodeRadius = 1;

int levelHeight = 0;

virtual void insertBalanced(char value);

void update();

void update(Node\* node, int col);

void draw();

void draw(Node\* node);

int getHeight();

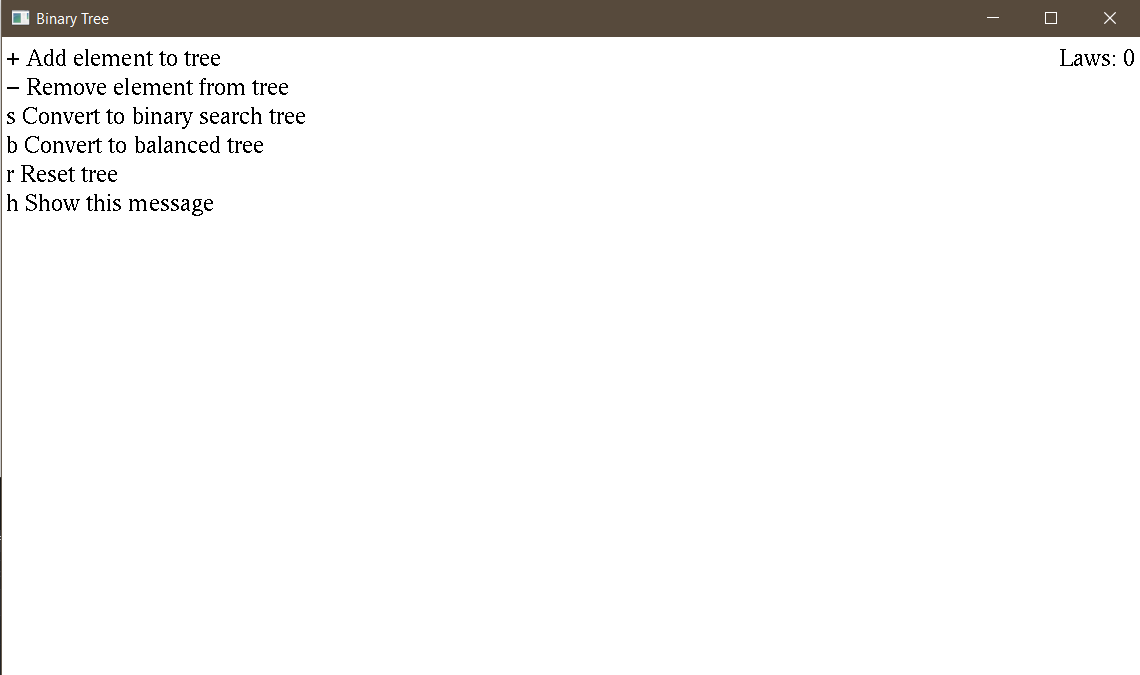
int getHeight(Node\* node);

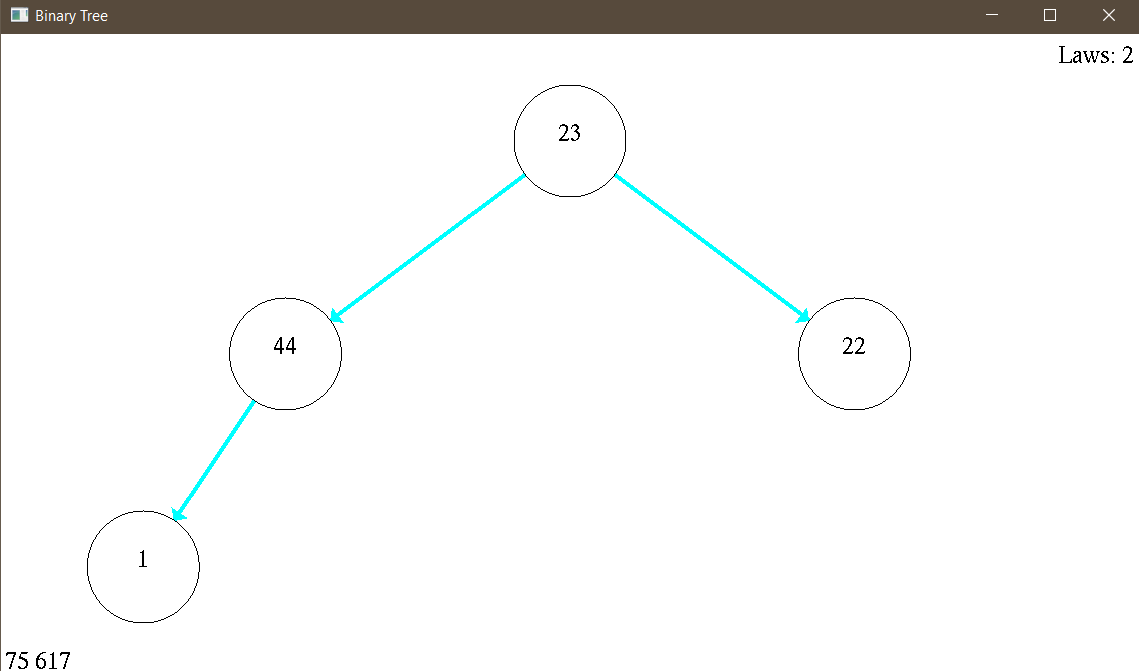
vector<char> getArray();

void reorder(Node\* node, char tmp);

};

**Скриншоты**

****

****