Министерство высшего образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет» (ПНИПУ)**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

Тема: «Бинарное дерево поиска»

Выполнил

Студент группы РИС-22-2б

Прядеин И.А.

Проверил доцент кафедры

ИТАС

Полякова О. А.

Пермь 2023

Постановка задачи

Реализовать алгоритмы бинарного дерева поиска, имеющего не менее трех уровней.

Алгоритмы:

1. Необходимо реализовать функции для редактирования дерева:

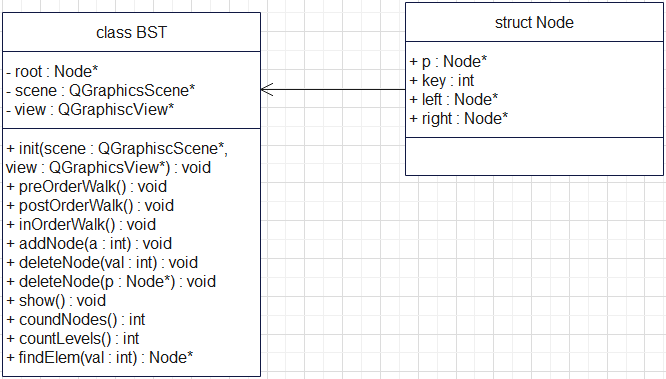
* Вставка узла
* Удаление узла
* Поиск элемента по ключу

1. Реализовать алгоритмы обхода дерева:

* Прямой
* Симметричный
* Обратный

1. Реализовать алгоритм балансировки дерева
2. Реализовать вертикальную и горизонтальную печать
3. Визуализация дерева должна быть выполнена с использованием любой доступной графической библиотеки - SFML, SDL, OpenGL и подобных.
4. Пользовательский интерфейс по усмотрению разработчика - с условием кроссплатформенности.

**UML-диаграмма**



**Код программы:**

**Файл bst.h:**

#ifndef BST\_H

#define BST\_H

#include <iostream>

#include <QGraphicsView>

#include <QTextStream>

#include <QProcess>

using namespace std;

struct **Node** {

Node \*p;

int key;

Node \*left;

Node \*right;

};

class **BST** {

public:

void **init**(QGraphicsScene\* scene, QGraphicsView\* view);

void **preOrderWalk**();

void **postOrderWalk**();

void **inOrderWalk**();

void **addNode**(int a);

void **deleteNode**(int val);

void **deleteNode**(Node\* p);

void **show**();

int **countNodes**();

int **countLevels**();

Node\* **findElem**(int val);

private:

int **countNodes**(Node\* p);

int **countLevels**(Node\* p);

void **preOrderWalk**(Node\* p);

void **postOrderWalk**(Node\* p);

void **inOrderWalk**(Node\* p);

Node\* **findSuccessor**(int val);

QByteArray **\_prepareGraph**();

void **\_graphWalk**(Node\* p, QTextStream\* stream);

Node\* **findElem**(int val, Node\* p);

Node\* root;

QGraphicsScene\* scene;

QGraphicsView\* view;

};

#endif // BST\_H

**Файл bst.cpp:**

#include "bst.h"

void BST::**init**(QGraphicsScene\* scene, QGraphicsView\* view) {

this->root = NULL;

this->scene = scene;

this->view = view;

}

void BST::**addNode**(int a) {

Node\* tmp = new Node;

tmp->key = a;

tmp->left = NULL;

tmp->right = NULL;

if (this->root == NULL) {

tmp->p = NULL;

this->root = tmp;

} else {

Node\* cElem = this->root;

Node\* parent = NULL;

while (cElem != NULL) {

parent = cElem;

cElem = (a < cElem->key) ? cElem->left : cElem->right;

}

tmp->p = parent;

if (a < parent->key)

parent->left = tmp;

else

parent->right = tmp;

}

}

void BST::**preOrderWalk**(Node\* p) {

if (p != NULL) {

cout << p->key << " ";

this->preOrderWalk(*p->left*);

this->preOrderWalk(*p->right*);

}

}

void BST::**preOrderWalk**() {

cout << "Preorder walk: ";

this->preOrderWalk(this*->root*);

cout << endl;

}

void BST::**postOrderWalk**(Node\* p) {

if (p != NULL) {

this->postOrderWalk(*p->left*);

this->postOrderWalk(*p->right*);

cout << p->key << " ";

}

}

void BST::**postOrderWalk**() {

cout << "Postorder walk: ";

this->postOrderWalk(this*->root*);

cout << endl;

}

void BST::**inOrderWalk**(Node\* p) {

if (p != NULL) {

this->inOrderWalk(*p->left*);

cout << p->key << " ";

this->inOrderWalk(*p->right*);

}

}

void BST::**inOrderWalk**() {

cout << "Inorder walk: ";

this->inOrderWalk(this*->root*);

cout << endl;

}

Node\* BST::**findElem**(int value, Node\* p) {

if (p != NULL) {

if (value == p->key)

return p;

if (value < p->key)

return findElem(value, *p->left*);

else

return findElem(value, *p->right*);

} else

return NULL;

}

Node\* BST::**findElem**(int value) {

return this->findElem(value, this*->root*);

}

Node\* BST::**findSuccessor**(int value) {

Node\* startNode = this->findElem(value);

Node\* parent = startNode;

startNode = startNode->right;

while (startNode != NULL && startNode->left != NULL){

parent = startNode;

startNode = startNode->left;

}

return startNode;

}

void BST::**deleteNode**(Node\* p) {

Node \*q = NULL;

Node \*r = NULL;

if (p->left == NULL || p->right == NULL)

q = p;

else

q = this->findSuccessor(p->key);

if (q->left != NULL)

r = q->left;

else

r = q->right;

if (r != NULL)

r->p = q->p;

if (q->p == NULL)

this->root = r;

else if (q == q->p->left)

q->p->left = r;

else

q->p->right = r;

if (q != p)

p->key = q->key;

}

void BST::**deleteNode**(int val) {

this->deleteNode(this*->findElem(val)*);

}

int BST::**countLevels**(Node\* p) {

int h1 = 0, h2 = 0;

if (p == NULL)

return 0;

if (p->left)

h1 = countLevels(*p->left*);

if (p->right)

h2 = countLevels(*p->right*);

return(max(h1,h2)+1);

}

int BST::**countLevels**() {

return this->countLevels(this*->root*);

}

int BST::**countNodes**(Node\* p) {

if(p == NULL)

return 0;

else

return (countNodes(*p->left*) + countNodes(*p->right*)+1);

}

int BST::**countNodes**() {

return this->countNodes(this*->root*);

}

void BST::**\_graphWalk**(Node\* p, QTextStream \*stream) {

if (p != NULL) {

\*stream << "\t\t" << "n" << p->key << "[label=\"" << p->key << "\"];" << endl;

if (p->left != NULL) {

\*stream << "\t\tn" << p->key << "--" << "n" << p->left->key << ";" << endl;

this->\_graphWalk(*p->left*, *stream*);

}

if (p->right != NULL) {

\*stream << "\t\tn" << p->key << "--" << "n" << p->right->key << ";" << endl;

this->\_graphWalk(*p->right*, *stream*);

}

}

}

QByteArray BST::**\_prepareGraph**() {

QByteArray arr = QByteArray();

QTextStream stream(*&arr*, QIODevice::ReadWrite);

stream << "graph ""{" << endl;

stream << "\tnode[fontsize=10,margin=0,width=\".4\", height=\".3\"];" << endl;

stream << "\tsubgraph cluster17{" << endl;

this->\_graphWalk(this*->root*, *&stream*);

stream << "\t}\n" << "}" << endl;

stream.flush();

return arr;

}

void BST::**show**() {

QProcess\* p = new QProcess();

QByteArray a = this->\_prepareGraph();

p->setProcessChannelMode(QProcess::MergedChannels);

p->start("dot", QStringList() << "-Tpng");

p->write(a);

QByteArray data;

QPixmap pixmap = QPixmap();

while(p->*waitForReadyRead*(100)) {

data.append(p->readAll());

}

pixmap.loadFromData(data);

this->scene->addPixmap(pixmap);

this->view->show();

}

**Файл main.cpp:**

#include "bst.h"

#include <QApplication>

using namespace std;

int main(int argc, char \*\*argv){

QApplication app(*argc*, *argv*);

QGraphicsScene scene;

QGraphicsView view(*&scene*);

view.setRenderHints(QPainter::SmoothPixmapTransform);

BST\* Tree = new BST();

Tree->init(*&scene*, *&view*);

Tree->addNode(7);

Tree->addNode(6);

Tree->addNode(10);

Tree->addNode(3);

Tree->addNode(4);

Tree->addNode(5);

Tree->addNode(8);

Tree->addNode(12);

Tree->addNode(9);

Tree->addNode(1);

Tree->addNode(2);

Tree->addNode(11);

cout << "Tree height: " << Tree->countLevels() << endl;

cout << "Number of nodes: " << Tree->countNodes() << endl;

cout << "Find element: " << Tree->findElem(12) << " : " << Tree->findElem(12)->key << endl;

Tree->preOrderWalk();

Tree->inOrderWalk();

Tree->postOrderWalk();

Tree->show();

return app.exec();

}

**Результат выполнения программы**

