ПНИПУ

Федеральное государственное автономное образовательное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

Лабораторная работа по теме

“Деревья. Графы. Алгоритм Дийкстры. Сбалансированное дерево поиска.”

Выполнил студент группы РИС-23-3Б

Буковский Денис Владимирович

Проверила доцент кафедры ИТАС О.А. Полякова

2024

Деревья

Постановка задачи

1. Сформировать идеально сбалансированное бинарное дерево, тип информационного поля указан в варианте.

2. Распечатать полученное дерево.

3. Выполнить обработку дерева в соответствии с заданием, вывести полученный результат.

4. Преобразовать идеально сбалансированное дерево в дерево поиска.

5. Распечатать полученное дерево.

Тип информационного поля int. Найти минимальный элемент в дереве.

1. Анализ задачи

Напишем реализацию бинарного дерева. Создадим OpenGL-ориентированный интерфейс в Qt Creator. Добавим визуализацию в приложение.

1. Блок-схема

\_

1. Код

BinaryTree.h

#ifndef BINARYTREE\_H

#define BINARYTREE\_H

#pragma once

#include <vector>

#include <queue>

#include <string>

*struct* **Tree** {

int data;

Tree\* left;

Tree\* right;

Tree\* parent;

**Tree**(int val, Tree\* par, Tree\* l, Tree\* r) {

data = val;

left = l;

right = r;

parent = par;

}

**Tree**(int val) {

*this*->data = val;

*this*->left = *nullptr*;

*this*->right = *nullptr*;

*this*->parent = *nullptr*;

}

**Tree**() {

data = -2000;

left = *nullptr*;

right = *nullptr*;

parent = *nullptr*;

}

Tree\* **minTree**(Tree\* tree) {

*if* (tree->left == *nullptr*) *return* *this*;

*return* tree->left->minTree(*tree->left*);

}

Tree\* **maxTree**(Tree\* tree) {

*if* (tree->right == *nullptr*) *return* *this*;

*return* tree->right->minTree(*tree->right*);

}

void **insert**(int val) {

Tree\* temp\_tree = *this*;

*while* (temp\_tree != *nullptr*) {

*if* (val > temp\_tree->data) {

*if* (temp\_tree->right != *nullptr*)

temp\_tree = temp\_tree->right;

*else* {

Tree\* tmp = *new* Tree(val);

tmp->parent = temp\_tree;

temp\_tree->right = tmp;

*break*;

}

}

*else* *if* (val < temp\_tree->data) {

*if* (temp\_tree->left != *nullptr*)

temp\_tree = temp\_tree->left;

*else* {

Tree\* tmp = *new* Tree(val);

tmp->parent = temp\_tree;

temp\_tree->left = tmp;

*break*;

}

}

*else*

*break*;

}

}

Tree\* **find**(int val) {

*if* (*this* == *nullptr*)

*return* *nullptr*;

*if* (*this*->data == val)

*return* *this*;

*else* *if* (val < *this*->data)

*return* *this*->left->find(val);

*else* *if* (val > *this*->data)

*return* *this*->right->find(val);

}

bool **erase**(int val){

Tree\* node = *this*->find(val);

*if* (node == *nullptr*) *return* *false*;

*if* ( (node->left == *nullptr*) && (node->right == *nullptr*) ) {

Tree\* node\_par = node->parent;

*if* (node\_par->left == node)

node->parent->left = *nullptr*;

*else*

node->parent->right = *nullptr*;

*delete* node;

}

*else* *if* ( (node->left == *nullptr* && node->right != *nullptr*) || (node->left != *nullptr* && node->right == *nullptr*) ) {

Tree\* node\_par = node->parent;

*if* (node->left == *nullptr*) {

*if* (node\_par->left == node)

node->parent->left = node->right;

*else*

node->parent->right = node->right;

node->right->parent = node->parent;

}

*else* {

*if* (node\_par->left == node)

node->parent->left = node->left;

*else*

node->parent->right = node->left;

node->left->parent = node->parent;

}

*delete* node;

}

*else* {

Tree\* r\_tree\_min = node->right->minTree(*node->right*);

*if* (r\_tree\_min->left == *nullptr* && r\_tree\_min->right == *nullptr*) {

int tmp = r\_tree\_min->data;

*this*->erase(r\_tree\_min->data);

node->data = tmp;

}

*else* {

int tmp = r\_tree\_min->data;

*this*->erase(r\_tree\_min->data);

node->data = tmp;

}

}

*return* *true*;

}

void **clear**() {

*if*(*this* != *nullptr*) {

*this*->left->clear();

*this*->right->clear();

*delete* *this*;

*return*;

}

*else*

*return*;

}

void **balance**() {

std::vector<int> values;

inorderTraversal(*this*, *values*);

Tree\* balancedTree = buildBalancedTree(values, 0, values.size() - 1);

\**this* = \*balancedTree;

}

void **inorderTraversal**(Tree\* node, std::vector<int>& values){

*if* (node == *nullptr*)

*return*;

inorderTraversal(*node->left*, *values*);

values.push\_back(node->data);

inorderTraversal(*node->right*, *values*);

}

Tree\* **buildBalancedTree**(*const* std::vector<int>& values, int start, int end) {

*if* (start > end)

*return* *nullptr*;

int mid = (start + end) / 2;

Tree\* newNode = *new* Tree(values[mid]);

newNode->left = buildBalancedTree(values, start, mid - 1);

*if* (newNode->left != *nullptr*)

newNode->left->parent = newNode;

newNode->right = buildBalancedTree(values, mid + 1, end);

*if* (newNode->right != *nullptr*)

newNode->right->parent = newNode;

*return* newNode;

}

};

*struct* **BinaryTree** {

Tree\* root;

**BinaryTree**() {

root = *nullptr*;

}

**BinaryTree**(int val) {

root = *new* Tree(val, *nullptr*, *nullptr*, *nullptr*);

}

void **insert**(int val) {

*this*->root->insert(val);

}

Tree\* **find**(int val) {

*return* (*this*->root->find(val));

}

bool **erase**(int val) {

*return* (*this*->root->erase(val));

}

void **balance**() {

*if*(*this*->root != *nullptr*)

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

}

void **clear**() {

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

*this*->root = *nullptr*;

}

};

#endif *//* *BINARYTREE\_H*

Mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QGraphicsScene>

#include <vector>

#include "BinaryTree.h"

QT\_BEGIN\_NAMESPACE

*namespace* **Ui** { *class* **MainWindow**; }

QT\_END\_NAMESPACE

*class* **MainWindow** : *public* QMainWindow

{

Q\_OBJECT

*public*:

**MainWindow**(QWidget \*parent = *nullptr*);

~***MainWindow***();

*private* slots:

void **on\_insert\_btn\_clicked**();

void **on\_search\_btn\_clicked**();

void **on\_del\_btn\_clicked**();

void **on\_pushButton\_clicked**();

void **on\_preOrder\_btn\_clicked**();

void **on\_del\_tree\_btn\_clicked**();

void **on\_simOrder\_btn\_clicked**();

void **on\_postOrder\_btn\_clicked**();

*private*:

Ui::MainWindow \*ui;

BinaryTree T;

QGraphicsScene\* **CreateScene**(Tree\*);

void **clear\_status**();

};

#endif *//* *MAINWINDOW\_H*

Mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QApplication>

#include <QGraphicsScene>

#include <QGraphicsView>

#include <QGraphicsEllipseItem>

#include <QGraphicsLineItem>

#include <cmath>

#include <string>

#include "BinaryTree.h"

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(*new* Ui::MainWindow)

{

ui->setupUi(*this*);

QGraphicsScene\* scene = CreateScene(*nullptr*);

ui->graphicsView->setScene(scene);

}

MainWindow::~MainWindow()

{

*delete* ui;

}

void preOrderTreeSceneCreate(Tree\* tree, QGraphicsScene\* Scene, int ell\_r, int lvlH, int lvlW, int lvlH\_delt, int lvlW\_delt, bool left, Tree\* to\_paint) {

*if*(tree != *nullptr*) {

int cur\_lvlW\_delt;

*if*(left) {

*if*(lvlW\_delt < 25) {

cur\_lvlW\_delt = lvlW\_delt + 50;

lvlW\_delt = 25;

}

*else*

cur\_lvlW\_delt = lvlW\_delt + 50;

}

*else* {

*if*(lvlW\_delt < 25){

cur\_lvlW\_delt = lvlW\_delt \* -1 - 50;

lvlW\_delt = 25;

}

*else*

cur\_lvlW\_delt = (lvlW\_delt + 50) \* -1;

}

*if*(tree->parent != *nullptr*) {

QGraphicsLineItem \*edge = Scene->addLine(lvlW, lvlH, lvlW + cur\_lvlW\_delt,lvlH - lvlH\_delt);

QPen PenEdge(Qt::black);

edge->setPen(PenEdge);

}

preOrderTreeSceneCreate(tree->left, Scene, ell\_r, lvlH + lvlH\_delt, lvlW - lvlW\_delt, lvlH\_delt, lvlW\_delt - 50, *true*, to\_paint);

preOrderTreeSceneCreate(tree->right, Scene, ell\_r, lvlH + lvlH\_delt, lvlW + lvlW\_delt, lvlH\_delt, lvlW\_delt - 50, *false*, to\_paint);

QGraphicsEllipseItem \*ELL = Scene->addEllipse(lvlW - ell\_r/2, lvlH - ell\_r/2, ell\_r, ell\_r);

QPen penELL(Qt::black);

QBrush brushELL(QColor(100, 153, 100));

*if*(to\_paint != *nullptr*) {

*if*(to\_paint->data == tree->data)

brushELL = QBrush(QColor(100, 100, 153));

}

ELL->setPen(penELL);

ELL->setBrush(brushELL);

int text\_delt = std::to\_string(tree->data).size();

QGraphicsTextItem \*text = Scene->addText(QString::number(tree->data));

text->setDefaultTextColor(Qt::black);

text->setPos(lvlW - 5 - 4 \* text\_delt, lvlH - 12);

}

*else* {

*return*;

}

}

QGraphicsScene\* MainWindow::CreateScene(Tree\* to\_paint) {

int ell\_r = 40;

int lvlH = 0;

int lvlW = 0;

int lvlH\_delt = 80;

int lvlW\_delt = 120;

QGraphicsScene\* new\_Scene = *new* QGraphicsScene;

preOrderTreeSceneCreate(T.root, new\_Scene, ell\_r, lvlH, lvlW, lvlH\_delt, lvlW\_delt, *false*, to\_paint);

*return* new\_Scene;

}

void preOrderQStringCreate(Tree\* tree, QString\* qstr) {

*if*(tree != *nullptr*) {

\*qstr += QString::number(tree->data);

\*qstr += " ";

preOrderQStringCreate(tree->left, qstr);

preOrderQStringCreate(tree->right, qstr);

}

*else*

*return*;

}

void simOrderQStringCreate(Tree\* tree, QString\* qstr) {

*if*(tree != *nullptr*) {

simOrderQStringCreate(tree->left, qstr);

\*qstr += QString::number(tree->data);

\*qstr += " ";

simOrderQStringCreate(tree->right, qstr);

}

*else*

*return*;

}

void postOrderQStringCreate(Tree\* tree, QString\* qstr) {

*if*(tree != *nullptr*) {

postOrderQStringCreate(tree->left, qstr);

postOrderQStringCreate(tree->right, qstr);

\*qstr += QString::number(tree->data);

\*qstr += " ";

}

*else*

*return*;

}

void MainWindow::clear\_status() {

ui->search\_status\_label->setText("");

ui->Order\_result\_textBrowser->setText("");

}

void MainWindow::on\_insert\_btn\_clicked() {

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

clear\_status();

int to\_add = ui->inser\_textEdit->toPlainText().toInt();

*if*(T.root == *nullptr*)

T.root = *new* Tree(to\_add);

*else*

T.insert(to\_add);

ui->inser\_textEdit->setText("");

QGraphicsScene \*new\_Scene = CreateScene(*nullptr*);

ui->graphicsView->setScene(new\_Scene);

*if*(prev\_scene) *delete* prev\_scene;

}

void MainWindow::on\_search\_btn\_clicked() {

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

clear\_status();

Tree\* found = T.find(ui->search\_textEdit->toPlainText().toInt());

QGraphicsScene \*new\_Scene;

*if*(found != *nullptr*) {

ui->search\_status\_label->setText("Элемент найден");

new\_Scene = CreateScene(found);

}

*else* {

ui->search\_status\_label->setText("Элемент не найден");

new\_Scene = CreateScene(*nullptr*);

}

ui->search\_textEdit->setText("");

ui->graphicsView->setScene(new\_Scene);

*if*(prev\_scene) *delete* prev\_scene;

}

void MainWindow::on\_del\_btn\_clicked() {

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

clear\_status();

int to\_del = ui->del\_textEdit->toPlainText().toInt();

bool flag = T.erase(to\_del);

ui->del\_textEdit->setText("");

QGraphicsScene \*new\_Scene = CreateScene(*nullptr*);

ui->graphicsView->setScene(new\_Scene);

*if*(prev\_scene) *delete* prev\_scene;

}

void MainWindow::on\_pushButton\_clicked() {

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

clear\_status();

T.balance();

QGraphicsScene \*new\_Scene = CreateScene(*nullptr*);

ui->graphicsView->setScene(new\_Scene);

*if*(prev\_scene) *delete* prev\_scene;

}

void MainWindow::on\_preOrder\_btn\_clicked() {

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

clear\_status();

QString RESULT;

preOrderQStringCreate(T.root, &RESULT);

ui->Order\_result\_textBrowser->setText(RESULT);

QGraphicsScene \*new\_Scene = CreateScene(*nullptr*);

ui->graphicsView->setScene(new\_Scene);

*if*(prev\_scene) *delete* prev\_scene;

}

void MainWindow::on\_del\_tree\_btn\_clicked() {

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

clear\_status();

T.clear();

QGraphicsScene \*new\_Scene = CreateScene(*nullptr*);

ui->graphicsView->setScene(new\_Scene);

*if*(prev\_scene) *delete* prev\_scene;

}

void MainWindow::on\_simOrder\_btn\_clicked() {

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

clear\_status();

QString RESULT;

simOrderQStringCreate(T.root, &RESULT);

ui->Order\_result\_textBrowser->setText(RESULT);

QGraphicsScene \*new\_Scene = CreateScene(*nullptr*);

ui->graphicsView->setScene(new\_Scene);

*if*(prev\_scene) *delete* prev\_scene;

}

void MainWindow::on\_postOrder\_btn\_clicked() {

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

clear\_status();

QString RESULT;

postOrderQStringCreate(T.root, &RESULT);

ui->Order\_result\_textBrowser->setText(RESULT);

QGraphicsScene \*new\_Scene = CreateScene(*nullptr*);

ui->graphicsView->setScene(new\_Scene);

*if*(prev\_scene) *delete* prev\_scene;

}

Main.cpp

#include "mainwindow.h"

#include <QApplication>

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

{

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

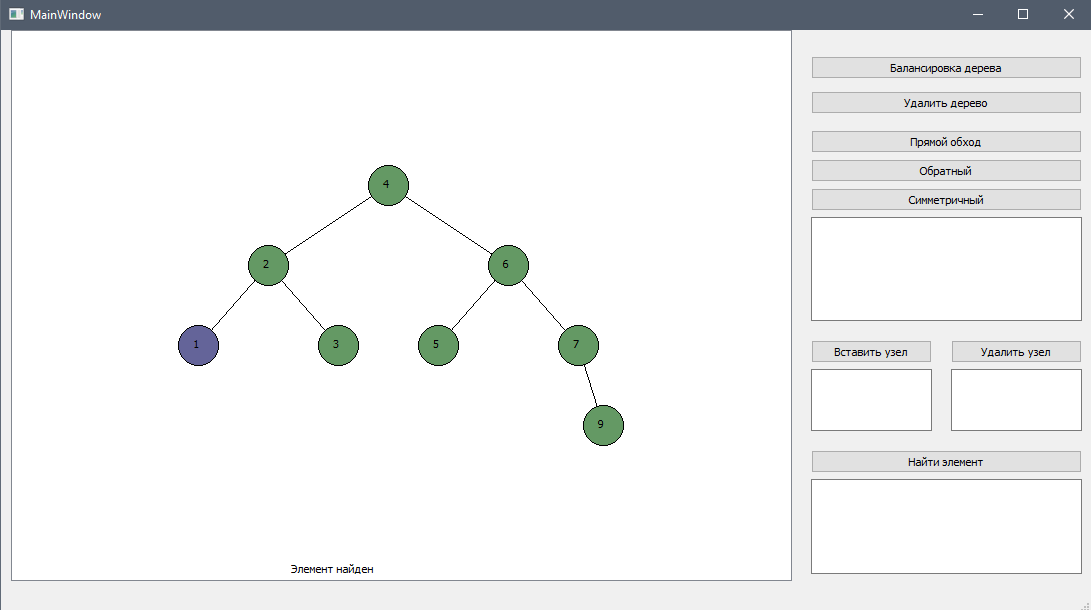
MainWindow w;

w.show();

*return* a.exec();

}

1. Вывод программы



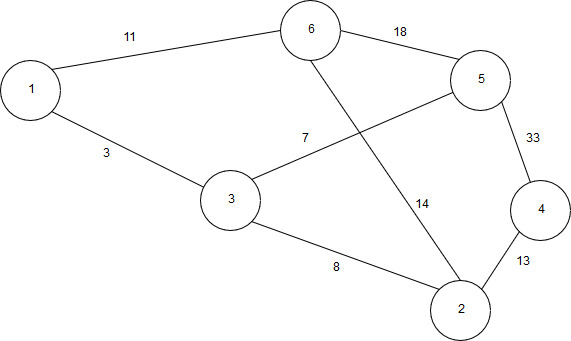
1. GitHub

<https://github.com/buksnet/OOPTree>

Графы. Алгоритм Дийкстры.

1. Постановка задачи

Реализовать граф, а также алгоритм Дейкстры, выполнив все необходимые действия. Выполнение начать с вершины 6.



1. Анализ задачи

Напишем реализацию графа. Создадим OpenGL-ориентированный интерфейс в Qt Creator. Добавим визуализацию в приложение.

1. Блок-схема

\_

1. Код

Adjacency.h

#ifndef ADJACENCY\_H

#define ADJACENCY\_H

#include <QWidget>

#include "graph.h"

*namespace* **Ui** {

*class* **Adjacency**;

}

*class* **Adjacency** : *public* QWidget

{

Q\_OBJECT

*public*:

*explicit* **Adjacency**(QWidget \*parent = *nullptr*);

~***Adjacency***();

void **setAdj**(QVector<QVector<int>>);

void **setSize**(int);

void **rebuild**(QVector<QString>);

void **setGraph**(Graph\*);

*private*:

Ui::Adjacency \*ui;

QVector<QVector<int>> adj;

int size;

Graph \*graph;

};

#endif *//* *ADJACENCY\_H*

Adjacency.cpp

#include "adjacency.h"

#include "ui\_adjacency.h"

#include <QDebug>

Adjacency::Adjacency(QWidget \*parent) :

QWidget(parent),

ui(*new* Ui::Adjacency)

{

ui->setupUi(*this*);

}

Adjacency::~Adjacency()

{

*delete* ui;

}

void Adjacency::setAdj(QVector<QVector<int>> adj)

{

*this*->adj = adj;

}

void Adjacency::setSize(int size)

{

*this*->size = size;

}

void Adjacency::rebuild(QVector<QString> verts)

{

QStringList names;

*for* (int i = 0; i < size; ++i ) {

names << verts[i];

}

ui->tableAdj->setRowCount(size);

ui->tableAdj->setColumnCount(size);

ui->tableAdj->setVerticalHeaderLabels(names);

ui->tableAdj->setHorizontalHeaderLabels(names);

*for* (int i=0; i<size; ++i){

*for*(int j=0; j<size; ++j){

QTableWidgetItem \*itm = *new* QTableWidgetItem(QString::number(adj[i][j]));

ui->tableAdj->setItem(i, j, itm);

ui->tableAdj->item(i, j)->setFlags(Qt::NoItemFlags);

}

}

}

void Adjacency::setGraph(Graph \* graph)

{

*this*->graph = graph;

}

Dialog.h

#ifndef DIALOG\_H

#define DIALOG\_H

#include "graphvertex.h"

#include "graph.h"

#include <adjacency.h>

#include <QDialog>

#include <QtCore>

#include <QtGui>

#include <QGraphicsScene>

*namespace* Ui {

*class* Dialog;

}

*class* Dialog : *public* QDialog

{

Q\_OBJECT

*public*:

Dialog(QWidget \*parent = *nullptr*);

~Dialog();

*private* *slots*:

void on\_btnAddVertex\_clicked();

void on\_btnAdj\_clicked();

void on\_btnAddEdge\_clicked();

void on\_btnDelete\_clicked();

void on\_btnTSP\_clicked();

void on\_btnSW\_clicked();

*private*:

Ui::Dialog \*ui;

Adjacency \*adj;

QGraphicsScene \* scene;

Graph graph;

graphVertex\* vert;

QVector<graphVertex\*> vertices;

Edge\* edge;

QVector<Edge\*> edges;

QQueue<graphVertex\*> for\_draw;

};

#endif *//* *DIALOG\_H*

Dialog.cpp

#include "dialog.h"

#include "ui\_dialog.h"

#include "graph.h"

#include <QDebug>

#include <QMouseEvent>

#include <QGraphicsEllipseItem>

Dialog::Dialog(QWidget \*parent) :

QDialog(parent),

ui(*new* Ui::Dialog())

{

ui->setupUi(*this*);

scene = *new* QGraphicsScene(*this*);

ui->graphicsView->setScene(scene);

}

Dialog::~Dialog()

{

*delete* ui;

}

void Dialog::on\_btnAddVertex\_clicked()

{

int num = vertices.size();

vert = *new* graphVertex(num);

vert->setGraph(&graph);

vertices.push\_back(vert);

scene->addItem(vert);

graph.insertVertex(num);

}

void Dialog::on\_btnAdj\_clicked()

{

adj = *new* Adjacency;

adj->setWindowTitle("Матрица смежности");

adj->setAdj(graph.getAdj());

adj->setSize(graph.getVertexList().size());

QVector<QString> names;

*for* (*auto* elem : graph.getVertexList())

{

names.push\_back(QString::number(elem));

}

adj->rebuild(names);

adj->show();

}

void Dialog::on\_btnAddEdge\_clicked()

{

graphVertex \*node1, \*node2;

int vert1, vert2;

*if* (ui->graphicsView->scene()->selectedItems().size() == 1) {

node1 = *dynamic\_cast*<graphVertex \*> (ui->graphicsView->scene()->selectedItems().at(0));

node1->setPressed(*true*);

*if* (for\_draw.isEmpty()) for\_draw.push\_back(node1);

*else* *if* (for\_draw.front() != node1 && for\_draw.front() != *nullptr*)

{

int result = ui->spinBox->value();

node2 = for\_draw.front();

vert1 = node1->getNum(); vert2 = node2->getNum();

graph.InsertEdge(vert2, vert1, result);

edge = *new* Edge(node1, node2);

edge->setGraph(&graph);

*if* (node1->pressed() == *true*) node1->setPressed(*false*);

*if* (node2->pressed() == *true*) node2->setPressed(*false*);

edge->setContent(result);

scene->addItem(edge);

edges.push\_back(edge);

for\_draw.pop\_back();

}

}

ui->graphicsView->scene()->clearSelection();

}

void Dialog::on\_btnDelete\_clicked()

{

*if* (ui->graphicsView->scene()->selectedItems().size()!=0)

{

int last = ui->graphicsView->scene()->selectedItems().size()-1;

*auto* current = ui->graphicsView->scene()->selectedItems().at(last);

*delete* current;

}

}

void Dialog::on\_btnSW\_clicked()

{

int current = 0;

current = ui->spinBox\_2->value();

*if* (current < graph.getVertexList().size())

{

QString res = "";

graph.ShortestWay(current, res);

ui->label->setText(res);

}

}

Edge.h

#ifndef EDGE\_H

#define EDGE\_H

#include <QGraphicsItem>

#include "graph.h"

*class* graphVertex;

*class* Edge : *public* QGraphicsItem

{

*public*:

Edge(graphVertex \*sourceVert, graphVertex \*destVert);

~Edge() override;

graphVertex \*sourceNode() *const*;

graphVertex \*destNode() *const*;

void setContent(int);

void adjust();

void setGraph(Graph\*);

*protected*:

QRectF boundingRect() *const* override;

void paint(QPainter \*painter, *const* QStyleOptionGraphicsItem \*option, QWidget \*widget) override;

*private*:

graphVertex \*source, \*dest;

bool \_pressed;

QPointF sourcePoint;

QPointF destPoint;

qreal arrowSize;

Graph \*graph;

QGraphicsItemGroup drawing;

QString content = "weight";

};

#endif *//* *EDGE\_H*

Edge.cpp

#include "edge.h"

#include "graphvertex.h"

#include <qmath.h>

#include <QPainter>

Edge::Edge(graphVertex \*sourceNode, graphVertex \*destNode)

: arrowSize(8)

{

setFlag(ItemIsSelectable);

source = sourceNode;

dest = destNode;

source->addEdge(*this*);

dest->addEdge(*this*);

adjust();

}

Edge::~Edge()

{

source->removeEdge(*this*);

dest->removeEdge(*this*);

int node1 = source->getNum(), node2 = dest->getNum();

graph->eraseEdge(node1, node2);

}

graphVertex \*Edge::sourceNode() *const*

{

*return* source;

}

graphVertex \*Edge::destNode() *const*

{

*return* dest;

}

void Edge::setContent(int num)

{

content = QString::number(num);

}

void Edge::adjust()

{

*if* (!source || !dest)

*return*;

QLineF line(mapFromItem(source, 0, 0), mapFromItem(dest, 0, 0));

qreal length = line.length();

prepareGeometryChange();

*if* (length > qreal(20.)) {

QPointF edgeOffset((line.dx() \* 10) / length, (line.dy() \* 10) / length);

sourcePoint = line.p1() + edgeOffset;

destPoint = line.p2() - edgeOffset;

} *else* {

sourcePoint = destPoint = line.p1();

}

}

void Edge::setGraph(Graph \*graph)

{

*this*->graph = graph;

}

QRectF Edge::boundingRect() *const*

{

*if* (!source || !dest)

*return* QRectF();

qreal penWidth = 3;

qreal extra = (penWidth + arrowSize) / 2.0;

*return* QRectF(sourcePoint,

QSizeF(destPoint.x() - sourcePoint.x(), destPoint.y() - sourcePoint.y()))

.normalized().adjusted(-extra, -extra, extra, extra);

}

void Edge::paint(QPainter \*painter, *const* QStyleOptionGraphicsItem \*, QWidget \*)

{

*if* (!source || !dest)

*return*;

QLineF line(sourcePoint, destPoint);

*if* (qFuzzyCompare(line.length(), qreal(0.)))

*return*;

painter->setPen(QPen(Qt::black, 2, Qt::SolidLine));

painter->drawLine(line);

painter->drawText((sourcePoint\*0.8+destPoint)/2, content);

double angle = std::atan2(-line.dy(), line.dx());

QPointF sourceArrowP1 = sourcePoint + QPointF(sin(angle + M\_PI / 3) \* arrowSize,

cos(angle + M\_PI / 3) \* arrowSize);

QPointF sourceArrowP2 = sourcePoint + QPointF(sin(angle + M\_PI - M\_PI / 3) \* arrowSize,

cos(angle + M\_PI - M\_PI / 3) \* arrowSize);

painter->setBrush(Qt::black);

painter->drawPolygon(QPolygonF() << line.p1() << sourceArrowP1 << sourceArrowP2);

}

Graph.h

#ifndef GRAPH\_H

#define GRAPH\_H

#include <QQueue>

#include <QVector>

#include <QDebug>

*class* Graph

{

*private*:

QVector<int> vertexList;

QVector<QVector<int>> adjMatrix;

int maxSize;

QQueue<int> VertsQueue;

QVector<int> labelList;

*public*:

Graph(*const* int& size);

Graph();

void showAdj();

QVector<QVector<int>> getAdj();

bool isEmpty();

bool isFull();

int GetVertPos(*const* int&);

int GetAmountVerts();

bool insertVertex(int&);

int GetAmountEdges();

int GetWeight(*const* int&, *const* int&);

int getWeightInd(*const* int&, *const* int&);

void eraseVertex(*const* int& vert);

void eraseEdge(*const* int& vert,*const* int& vert2);

void InsertEdge(*const* int& vertex1, *const* int& vertex2, *const* int& weight = 1);

QVector<int> GetNbrs(*const* int&);

QVector<int> getVertexList();

bool AllVisited(QVector<bool> &visitedVerts);

void FillLabels(int &startVertex);

void ShortestWay(int &startVertex, QString &res);

};

#endif *//* *GRAPH\_H*

Graph.cpp

#include "graph.h"

Graph::Graph(*const* int &size)

{

*this*->labelList = QVector<int>(size);

*this*->maxSize = size;

*this*->adjMatrix = QVector<QVector<int>>(size, QVector<int>(size));

*for* (int i = 0; i < *this*->maxSize; ++i)

{

*for* (int j = 0; j < *this*->maxSize; ++j) *this*->adjMatrix[i][j] = 0;

};

}

Graph::Graph() : Graph(10) {}

*inline* bool Graph::isEmpty() { *return* *this*->vertexList.size() == 0; };

*inline* bool Graph::isFull() { *return* *this*->vertexList.size() == *this*->maxSize;};

*inline* int Graph::GetVertPos(*const* int& v)

{

*for* (int i = 0; i < *this*->vertexList.size(); ++i)

{

*if* (*this*->vertexList[i] == v) *return* i;

}

*return* -1;

};

*inline* int Graph::GetAmountVerts() { *return* *this*->vertexList.size(); };

bool Graph::insertVertex(int& vert)

{

*if* (*this*->isFull())

{

qDebug() << "Невозможно добавить вершину.";

*return* *false*;

};

*if* (*this*->GetVertPos(vert) == -1) {

*this*->vertexList.push\_back(vert);

*return* *true*;

}

qDebug() << vertexList;

*return* *false*;

};

int Graph::GetAmountEdges()

{

int amount = 0;

*if* (!*this*->isEmpty()){

*for* (int i = 0; i < *this*->vertexList.size(); ++i) {

*for* (int j = 0; j < *this*->vertexList.size(); ++j) *if* (*this*->adjMatrix[i][j] != 0) amount++;

}

}

*return* amount;

};

*inline* int Graph::GetWeight(*const* int& v1, *const* int& v2)

{

*if* (*this*->isEmpty()) *return* 0;

int v1\_p = *this*->GetVertPos(v1);

int v2\_p = *this*->GetVertPos(v2);

*if* (v1\_p == -1 || v2\_p == -1) {

qDebug() << "Одного из узлов в графе не существует." << endl;

*return* 0;

};

*return* *this*->adjMatrix[v1\_p][v2\_p];

};

QVector<int> Graph::GetNbrs(*const* int& vertex)

{

QVector<int> nbrsList;

int pos = *this*->GetVertPos(vertex);

*if* (pos != -1)

{

*for* (int i = 0; i < *this*->vertexList.size(); ++i)

{

*if* (*this*->adjMatrix[pos][i] != 0) nbrsList.push\_back(*this*->vertexList[i]);

}

}

*return* nbrsList;

}

QVector<int> Graph::getVertexList()

{

*return* *this*->vertexList;

};

void Graph::InsertEdge(*const* int& vertex1, *const* int& vertex2, *const* int& weight)

{

*if* (GetVertPos(vertex1) != (-1) && *this*->GetVertPos(vertex2) != (-1))

{

int vertPos1 = GetVertPos(vertex1);

int vertPos2 = GetVertPos(vertex2);

*if* (*this*->adjMatrix[vertPos1][vertPos2] != 0)

{

qDebug() << "Ребро между вершинами уже есть" << endl;

*return*;

}

*else* *this*->adjMatrix[vertPos1][vertPos2] = weight;

}

*else*

{

qDebug() << "Обеих вершин (или одной из них) нет в графе " << endl;

*return*;

}

};

void Graph::showAdj()

{

*for* (int i = 0; i < *this*->maxSize; ++i) qDebug() << *this*->adjMatrix[i] << ' ';

qDebug() << endl;

}

QVector<QVector<int> > Graph::getAdj()

{

*return* *this*->adjMatrix;

};

void Graph::eraseEdge(*const* int& vert, *const* int& vert2)

{

int p1 = *this*->GetVertPos(vert);

int p2 = *this*->GetVertPos(vert2);

*if* (p1 != -1 && p2 != -1)

{

*this*->adjMatrix[p1][p2] = 0;

}

};

void Graph::eraseVertex(*const* int& vert)

{

int pos = *this*->GetVertPos(vert);

*if* ( pos != -1)

{

*for* (int i = 0; i < *this*->vertexList.size(); ++i)

{

*this*->adjMatrix[i].erase(*this*->adjMatrix[i].begin() + pos);

}

*this*->adjMatrix.erase(*this*->adjMatrix.begin() + pos);

*this*->vertexList.erase(*this*->vertexList.begin() + pos);

}

};

bool Graph::AllVisited(QVector<bool> &visitedVerts)

{

bool flag = *true*;

*for* (int i = 0; i < *this*->vertexList.size(); i++)

*if* (visitedVerts[i] != *true*)

flag = *false*;

*return* flag;

}

void Graph::FillLabels(int &startVertex)

{

*for* (int i = 0, size = vertexList.size(); i < size; ++i)

labelList[i] = 1000000;

int pos = GetVertPos(startVertex);

labelList[pos] = 0;

}

void Graph::ShortestWay(int &startVertex, QString &res)

{

*for* (int i = 0; i < vertexList.size(); i++)

*for* (int j = 0; j < vertexList.size(); j++)

*if* (adjMatrix[i][j] < 0)

*return*;

*if* (GetVertPos(startVertex) == -1)

*return*;

QVector<bool> visitedVerts(vertexList.size());

*for* (*auto* it = visitedVerts.begin(); it!=visitedVerts.end(); ++it) \*it = *false*;

*this*->FillLabels(startVertex);

int curSrc = startVertex;

QVector<int> neighbors;

*while* (!*this*->AllVisited(visitedVerts))

{

neighbors = *this*->GetNbrs(curSrc);

int startLabel = labelList[GetVertPos(curSrc)];

int\* minNeighbor\_ptr = *nullptr*;

int minW = 1000000;

*for* (int i = 0; i < neighbors.size(); ++i)

{

int weight = *this*->GetWeight(curSrc, neighbors[i]);

int nIndex = *this*->GetVertPos(neighbors[i]);

int nextLabel = labelList[nIndex];

*if* (startLabel + weight < nextLabel) labelList[nIndex] = startLabel + weight;

*if* (!visitedVerts[nIndex] && minW > labelList[nIndex])

{

minW = labelList[nIndex];

minNeighbor\_ptr = &neighbors[i];

}

}

visitedVerts[GetVertPos(curSrc)] = *true*;

*if* (minNeighbor\_ptr != *nullptr*) curSrc = \*minNeighbor\_ptr;

}

*for* (int i = 0; i < vertexList.size(); ++i)

{

*if* (i != GetVertPos(startVertex))

{

res+= QString::number(startVertex)

+ " -> " + QString::number(vertexList[i]) + " = "

+ QString::number(labelList[GetVertPos(vertexList[i])]) + "\n";

}

}

}

Graphvertex.h

#ifndef GRAPHVERTEX\_H

#define GRAPHVERTEX\_H

#include <QPainter>

#include <QGraphicsItem>

#include <QGraphicsEllipseItem>

#include <QGraphicsScene>

#include "edge.h"

*static* *const* int RADIUS = 30;

*class* **graphVertex** : *public* QGraphicsItem

{

*friend* Edge::**Edge**(graphVertex \*sourceVert, graphVertex \*destVert);

*friend* Edge::~***Edge***();

*public*:

**graphVertex**(int);

~***graphVertex***();

QList<Edge \*> **edges**() *const*;

QRectF ***boundingRect***() *const* *override*;

QPainterPath ***shape***() *const* *override*;

void ***paint***(QPainter \* painter, *const* QStyleOptionGraphicsItem \*option, QWidget\* widget) *override*;

bool **pressed**() *const*;

void **setPressed**(bool pressed);

QString **getContent**();

void **setGraph**(Graph\*);

int **getNum**();

*protected*:

QVariant ***itemChange***(GraphicsItemChange change, *const* QVariant &value) *override*;

void ***mousePressEvent***(QGraphicsSceneMouseEvent \*event) *override*;

void ***mouseReleaseEvent***(QGraphicsSceneMouseEvent \*event) *override*;

void **addEdge**(Edge \*edge);

void **removeEdge**(Edge \*edge);

*private*:

QList<Edge \*> edgeList;

bool \_pressed;

QGraphicsItemGroup \*drawing;

Graph \*graph;

int num = 0;

QString content = " ";

};

#endif *//* *GRAPHVERTEX\_H*

Graphvertex.cpp

#include "graphvertex.h"

#include <QFontDatabase>

graphVertex::**graphVertex**(int num)

{

*this*->num = num;

\_pressed = *false*;

setFlag(*ItemIsSelectable*);

setFlag(*ItemIsMovable*);

setFlag(*ItemSendsGeometryChanges*);

drawing = *new* QGraphicsItemGroup;

}

graphVertex::~***graphVertex***()

{

*while* (!edgeList.isEmpty())

*delete* edgeList.takeFirst();

graph->eraseVertex(num);

}

QRectF graphVertex::***boundingRect***() *const*

{

*return* *shape*().boundingRect();

}

bool graphVertex::**pressed**() *const*

{

*return* \_pressed;

}

void graphVertex::**setPressed**(bool pressed)

{

\_pressed = pressed;

update();

}

QString graphVertex::**getContent**()

{

*return* content;

}

void graphVertex::**setGraph**(Graph \*graph)

{

*this*->graph = graph;

}

int graphVertex::**getNum**()

{

*return* num;

}

QPainterPath graphVertex::***shape***() *const*

{

QPainterPath path;

path.addEllipse(-RADIUS, -RADIUS, 2 \* RADIUS, 2 \* RADIUS);

*return* path;

}

void graphVertex::***paint***(QPainter \*painter, *const* QStyleOptionGraphicsItem \*option, QWidget\* widget)

{

QBrush brush(Qt::*blue*);

*if* (pressed()) brush.setColor(QColor(200, 200, 200));

*else* brush.setColor(QColor(255, 255, 255));

painter->setBrush(brush);

painter->drawEllipse(-RADIUS, -RADIUS, 2 \* RADIUS, 2 \* RADIUS);

content = QString::number(num);

painter->drawText(*boundingRect*(), Qt::*AlignCenter*, content);

}

QVariant graphVertex::***itemChange***(GraphicsItemChange change, *const* QVariant &value)

{

*switch* (change)

{

*case* *ItemPositionHasChanged*:

foreach (Edge \*edge, edgeList)

edge->adjust();

*break*;

*default*:

*break*;

};

*return* QGraphicsItem::itemChange(change, value);

}

void graphVertex::***mousePressEvent***(QGraphicsSceneMouseEvent \*event)

{

\_pressed = *true*;

update();

QGraphicsItem::mousePressEvent(*event*);

}

void graphVertex::***mouseReleaseEvent***(QGraphicsSceneMouseEvent\* event)

{

\_pressed = *false*;

update();

QGraphicsItem::mouseReleaseEvent(*event*);

}

void graphVertex::**addEdge**(Edge \*edge)

{

edgeList << edge;

edge->adjust();

}

void graphVertex::**removeEdge**(Edge \*edge)

{

edgeList.removeOne(edge);

}

Mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

QT\_BEGIN\_NAMESPACE

*namespace* **Ui** { *class* **MainWindow**; }

QT\_END\_NAMESPACE

*class* **MainWindow** : *public* QMainWindow

{

Q\_OBJECT

*public*:

**MainWindow**(QWidget \*parent = *nullptr*);

~***MainWindow***();

*private*:

Ui::MainWindow \*ui;

};

#endif *//* *MAINWINDOW\_H*

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;

}

Main.cpp

#include "dialog.h"

#include <QApplication>

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

{

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

Dialog w;

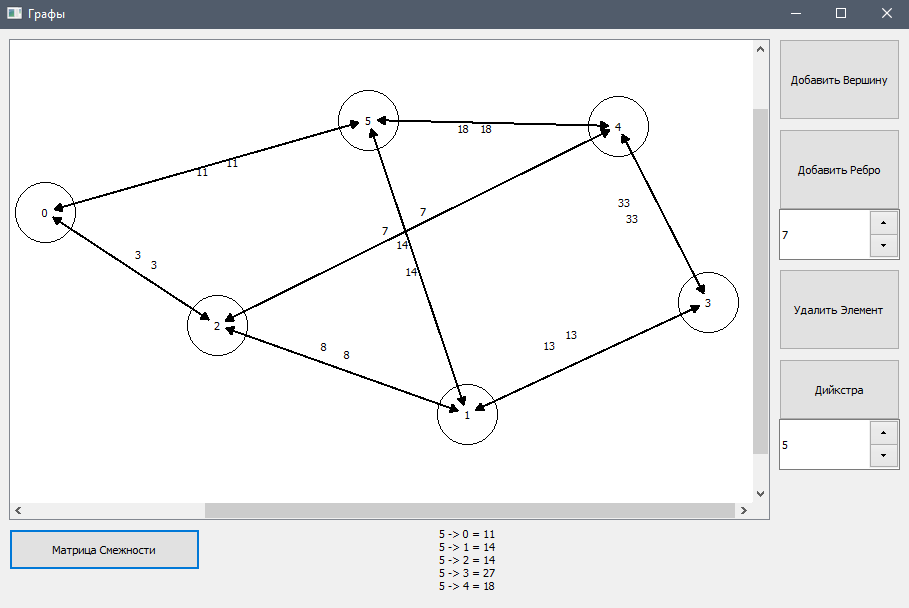
w.setWindowFlags (w.windowFlags() & ~Qt::*WindowContextHelpButtonHint* & Qt::*WindowMaximizeButtonHint*);

w.setWindowTitle("Графы");

w.show();

*return* a.exec();

}

1. Вывод программы
2. 
3. GitHub

https://github.com/buksnet/OOPGraph