**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Математического обеспечения и применения ЭВМ**

отчет

**по лабораторной работе № 2**

**по дисциплине «Построение и Анализ Алгоритмов»**

Тема: Нахождение компонент двусвязности.

|  |  |  |
| --- | --- | --- |
| Студент гр. 6304 |  | Ковынев М.В. |
| Преподаватель |  | Балтрашевич В.Э. |

Санкт-Петербург

2018

# **Цель работы**

Реализовать алгоритм нахождение компонент двусвязности в неориентированном графе.

# **Ход работы**

1. Реализуем функцию класс, в котором будет храниться подграф с V вершнами и матрицей смежности adj:

class Graph

{

int V;

list<int> \*adj;

bool **isBCUtil**(int v, bool visited[], int disc[], int low[],

int parent[]);

public:

Graph(int V);

void **addEdge**(int v, int w);

bool **isBC**();

};

1. Реализуем функцию проверки данного подграфа на двусвязность:

bool Graph::**isBCUtil**(int u, bool visited[], int disc[],int low[],int parent[])

{

static int time = 0;

int children = 0;

visited[u] = true;

disc[u] = low[u] = ++time;

list<int>::iterator i;

for (i = adj[u].begin(); i != adj[u].end(); ++i)

{

int v = \*i;

if (!visited[v])

{

children++;

parent[v] = u;

if (isBCUtil(v, visited, disc, low, parent))

return true;

low[u] = min(low[u], low[v]);

if (parent[u] == NIL && children > 1)

return true;

if (parent[u] != NIL && low[v] >= disc[u])

return true;

}

else

if (v != parent[u])

low[u] = min(low[u], disc[v]);

}

return false;

}

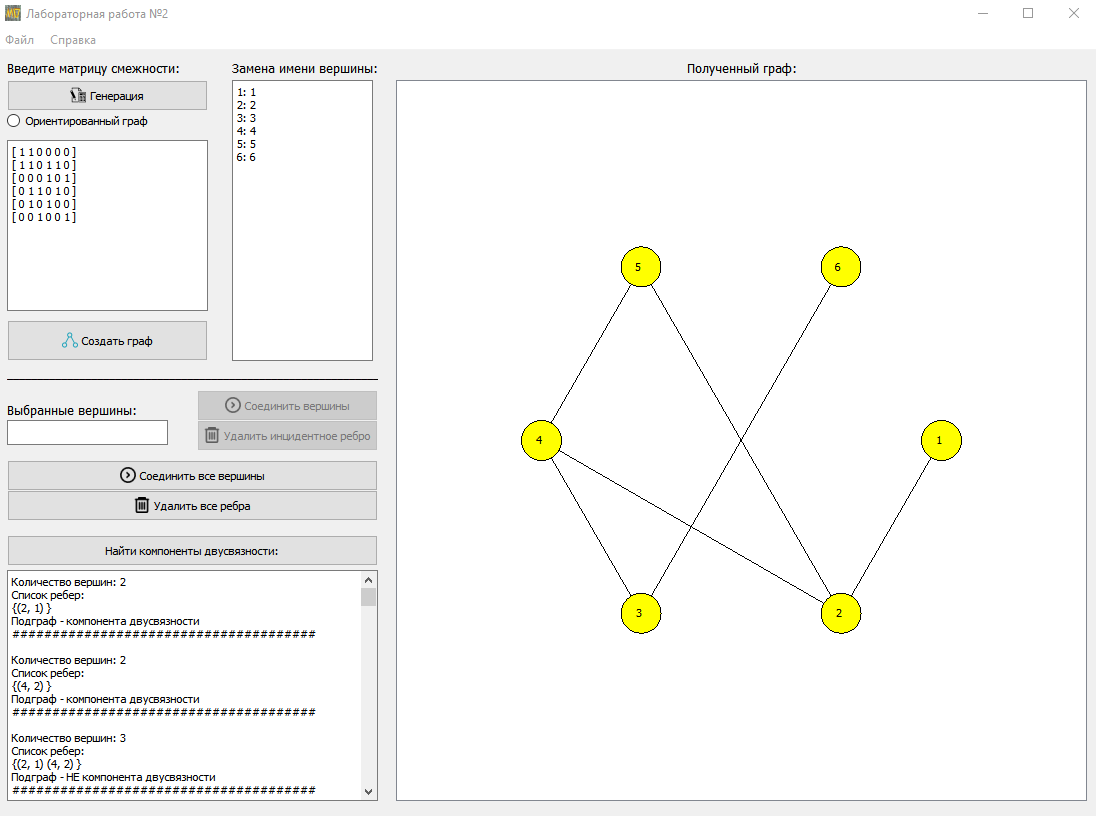


Рисунок 1. Интерфейс программы.

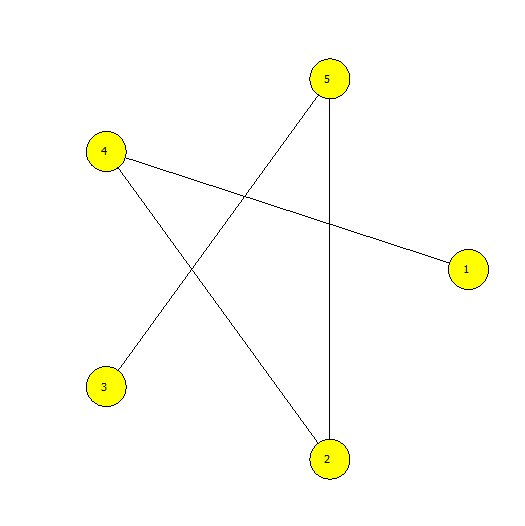


Рисунок 2. Пример построенного графа.

Результат обхода всех подграфов для рис. 2:

Количество вершин: 2

Список ребер:

{(4, 1) }

Подграф - компонента двусвязности

######################################

Количество вершин: 2

Список ребер:

{(4, 2) }

Подграф - компонента двусвязности

######################################

Количество вершин: 3

Список ребер:

{(4, 1) (4, 2) }

Подграф - НЕ компонента двусвязности

######################################

Количество вершин: 2

Список ребер:

{(5, 2) }

Подграф - компонента двусвязности

######################################

Количество вершин: 3

Список ребер:

{(4, 1) (5, 2) }

Подграф - НЕ компонента двусвязности

######################################

Количество вершин: 3

Список ребер:

{(4, 2) (5, 2) }

Подграф - НЕ компонента двусвязности

######################################

Количество вершин: 4

Список ребер:

{(4, 1) (4, 2) (5, 2) }

Подграф - НЕ компонента двусвязности

######################################

Количество вершин: 2

Список ребер:

{(5, 3) }

Подграф - компонента двусвязности

######################################

Количество вершин: 3

Список ребер:

{(4, 1) (5, 3) }

Подграф - НЕ компонента двусвязности

######################################

Количество вершин: 3

Список ребер:

{(4, 2) (5, 3) }

Подграф - НЕ компонента двусвязности

######################################

Количество вершин: 4

Список ребер:

{(4, 1) (4, 2) (5, 3) }

Подграф - НЕ компонента двусвязности

######################################

Количество вершин: 3

Список ребер:

{(5, 2) (5, 3) }

Подграф - НЕ компонента двусвязности

######################################

Количество вершин: 4

Список ребер:

{(4, 1) (5, 2) (5, 3) }

Подграф - НЕ компонента двусвязности

######################################

Количество вершин: 4

Список ребер:

{(4, 2) (5, 2) (5, 3) }

Подграф - НЕ компонента двусвязности

######################################

Количество вершин: 5

Список ребер:

{(4, 1) (4, 2) (5, 2) (5, 3) }

Подграф - НЕ компонента двусвязности

######################################

# **Вывод**

В ходе выполнения данной лабораторной работы были изучены основные понятия и приёмы применения алгоритма нахождения компонент двусвязнгости для решения поставленной задачи.

# **ПРИЛОЖЕНИЕ А.**

# **ИХОДНЫЙ КОД BICONNECTED.H**

#include<iostream>

#include <list>

#define NIL -1

using namespace std;

class Graph

{

int V;

list<int> \*adj;

bool **isBCUtil**(int v, bool visited[], int disc[], int low[],

int parent[]);

public:

Graph(int V);

void **addEdge**(int v, int w);

bool **isBC**();

};

# **ПРИЛОЖЕНИЕ Б.**

# **ИХОДНЫЙ КОД BICONNECTED.CPP**

#include "biconnected.h"

#define NIL -1

using namespace std;

Graph::**Graph**(int V)

{

this->V = V;

adj = new list<int>[V];

}

void Graph::**addEdge**(int v, int w)

{

adj[v].push\_back(w);

adj[w].push\_back(v);

}

bool Graph::**isBCUtil**(int u, bool visited[], int disc[],int low[],int parent[])

{

static int time = 0;

int children = 0;

visited[u] = true;

disc[u] = low[u] = ++time;

list<int>::iterator i;

for (i = adj[u].begin(); i != adj[u].end(); ++i)

{

int v = \*i;

if (!visited[v])

{

children++;

parent[v] = u;

if (isBCUtil(v, visited, disc, low, parent))

return true;

low[u] = min(low[u], low[v]);

if (parent[u] == NIL && children > 1)

return true;

if (parent[u] != NIL && low[v] >= disc[u])

return true;

}

else

if (v != parent[u])

low[u] = min(low[u], disc[v]);

}

return false;

}

bool Graph::**isBC**()

{

bool \*visited = new bool[V];

int \*disc = new int[V];

int \*low = new int[V];

int \*parent = new int[V];

for (int i = 0; i < V; i++)

{

parent[i] = NIL;

visited[i] = false;

}

if (isBCUtil(0, visited, disc, low, parent) == true)

return false;

for (int i = 0; i < V; i++)

if (visited[i] == false)

return false;

return true;

}

# **ПРИЛОЖЕНИЕ В.**

# **ИХОДНЫЙ EDGE.H**

#ifndef EDGE\_H

#define EDGE\_H

#include <QGraphicsItem>

class Node;

class Edge : public QGraphicsItem

{

public:

Edge(Node \*sourceNode, Node \*destNode, int direction);

Node \***sourceNode**() const;

Node \***destNode**() const;

int direction;

void **adjust**();

enum { Type = UserType + 2 };

int ***type***() const override { return Type; }

protected:

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

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

private:

Node \*source, \*dest;

QPointF sourcePoint;

QPointF destPoint;

qreal arrowSize;

};

#endif // EDGE\_H

# **ПРИЛОЖЕНИЕ Г.**

# **ИХОДНЫЙ КОД EDGE.CPP**

#include "edge.h"

#include "node.h"

#include <qmath.h>

#include <QPainter>

Edge::**Edge**(Node \*sourceNode, Node \*destNode, int direction1)

: arrowSize(10)

{

setAcceptedMouseButtons(0);

source = sourceNode;

dest = destNode;

source->addEdge(this);

dest->addEdge(this);

adjust();

direction = direction1;

}

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

{

return source;

}

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

{

return dest;

}

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() +2\* edgeOffset;

destPoint = line.p2() - 2\*edgeOffset;

}

else

{

sourcePoint = destPoint = line.p1();

}

}

QRectF Edge::***boundingRect***() const

{

if (!source || !dest)

return QRectF();

qreal penWidth = 1;

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, 1, Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin));

painter->drawLine(line);

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

QPointF sourceArrowP1;

QPointF sourceArrowP2;

painter->setBrush(Qt::black);

if (direction == 1)

{

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

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

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

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

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

}

}

# **ПРИЛОЖЕНИЕ Д.**

# **ИХОДНЫЙ КОД GRAPHWIDGET.H**

#ifndef GRAPHWIDGET\_H

#define GRAPHWIDGET\_H

#include <QGraphicsView>

#include <QTextEdit>

#include <algorithm>

#include <QPushButton>

#include <QMessageBox>

#include <QFile>

#include <QFileDialog>

#include <QTextStream>

#include <QMainWindow>

#include <vector>

#include <map>

using namespace std;

class Node;

struct MyNode

{

QString name;

int value;

QPointF pos;

};

class GraphWidget : public QGraphicsView

{

Q\_OBJECT

public:

GraphWidget(vector<vector<MyNode>>, QTextEdit\* , int, QWidget \*parent = 0);

void **itemMoved**();

protected:

void ***timerEvent***(QTimerEvent \*event) override;

private:

int timerId;

};

#endif // GRAPHWIDGET\_H

# **ПРИЛОЖЕНИЕ Е.**

# **ИХОДНЫЙ КОД GRAPHWIDGET.CPP**

#include "graphwidget.h"

#include "edge.h"

#include "node.h"

#include <math.h>

#include <QRandomGenerator>

#include <iostream>

#include <vector>

#include <QPointF>

using namespace std;

GraphWidget::**GraphWidget**(vector<vector<MyNode>> a, QTextEdit\* textEdit, int flag, QWidget \*parent)

: QGraphicsView(parent), timerId(0)

{

QGraphicsScene \*scene = new QGraphicsScene(this);

scene->setItemIndexMethod(QGraphicsScene::NoIndex);

scene->setSceneRect(0, 0, 600, 600);

setScene(scene);

vector<Node\*> array(a[0].size());

for (unsigned int i=0; i<array.size(); i++)

{

array[i] = new Node(a[i][i].name, textEdit, a[i][i].pos, this);

array[i]->setPos(a[i][i].pos);

scene->addItem(array[i]);

}

for (unsigned int i=0; i < a[0].size(); i++)

{

for (unsigned int j=0; j< a[0].size(); j++)

{

if (a[i][j].value != 0)

{

scene->addItem(new Edge(array[j], array[i], flag));

}

}

}

}

void GraphWidget::**itemMoved**()

{

if (!timerId)

timerId = startTimer(1000 / 25);

}

void GraphWidget::***timerEvent***(QTimerEvent \*event)

{

Q\_UNUSED(event);

QList<Node\*> nodes;

foreach (QGraphicsItem \*item, scene()->items())

{

if (Node \*node = qgraphicsitem\_cast<Node \*>(item))

nodes << node;

}

}

# **ПРИЛОЖЕНИЕ Ж.**

# **ИХОДНЫЙ КОД MAINWIDNOW.H**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include "graphwidget.h"

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~***MainWindow***();

private slots:

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

void **on\_textEdit\_textChanged**();

void **on\_pushButton\_2\_clicked**();

vector<vector<MyNode>> **textEditToVector**();

void **on\_pushButton\_3\_clicked**();

void **updateMatrix**(const vector<vector<MyNode>>& );

void **on\_pushButton\_4\_clicked**();

void **on\_pushButton\_5\_clicked**();

void **on\_pushButton\_6\_clicked**();

void **on\_action\_triggered**();

void **on\_action\_2\_triggered**();

void **on\_action\_4\_triggered**();

void **on\_action\_3\_triggered**();

void **on\_pushButton\_7\_clicked**();

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

private:

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

# **ПРИЛОЖЕНИЕ И.**

# **ИХОДНЫЙ КОД MAINWIDNOW.CPP**

#include "mainwindow.h"

#include "biconnected.h"

#include "ui\_mainwindow.h"

#include "graphwidget.h"

double PI = 3.14159265358979323846264338327950288419717;

int flag = 0;

MainWindow::**MainWindow**(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

ui->pushButton\_3->setDisabled(1);

ui->pushButton\_5->setDisabled(1);

}

MainWindow::~***MainWindow***()

{

delete ui;

}

// Text to Vector

vector<vector<MyNode>> MainWindow::**textEditToVector**()

{

QStringList namesList = ui->textEdit\_3->toPlainText()

.replace(":", "\n").replace("\n", " ").split(" ");

vector<QString> vector\_names;

for (int i = 2; i< namesList.size(); i = i+3)

{

vector\_names.push\_back(namesList[i]);

}

QString str = ui->textEdit\_2->toPlainText().replace("\n", " ");

int size = count(begin(str), end(str), "[");

vector<vector<MyNode>> matrix(1, vector<MyNode>(0));

if (size == 0)

{

return matrix;

}

QStringList list = str.split(' ');

int j = 0;

int k = 0 ;

matrix.resize(size);

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

matrix[i].resize(size);

double angle = 2 \* PI / matrix[0].size();

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

{

if (list[i] != "[" && list[i] != "")

{

if (list[i] != "]")

{

QString x = list[i];

matrix[j][k].value = x.toInt();

matrix[j][k].name = vector\_names[k];

matrix[j][k].pos.setX(300 + 200\*cos(angle \* k));

matrix[j][k].pos.setY(300 + 200\*sin(angle \* k));

k++;

}

else

{

j++;

k=0;

}

}

}

return matrix;

}

// Draw graph

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

{

ui->textEdit\_4->clear();

vector<vector<MyNode>> matrix = textEditToVector();

if (matrix[0].size() != 0)

{

GraphWidget \*widget = new GraphWidget(matrix, ui->textEdit, flag, this);

ui->graphicsView->setScene(widget->scene());

}

else

{

ui->textEdit\_2->setPlaceholderText("Матрица некорректна!\nВведите другие данные!");

}

}

// Get elements with double click

void MainWindow::**on\_textEdit\_textChanged**()

{

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

QStringList list = str.split(" ");

int count = list.size() - 1;

if (count > 2)

{

ui->textEdit->clear();

ui->pushButton\_3->setDisabled(1);

ui->pushButton\_5->setDisabled(1);

}

else

if (count == 2)

{

ui->pushButton\_3->setEnabled(1);

ui->pushButton\_5->setEnabled(1);

}

else

if (count < 2)

{

ui->pushButton\_3->setDisabled(1);

ui->pushButton\_5->setDisabled(1);

}

}

// Generate TextEdit

void MainWindow::**on\_pushButton\_2\_clicked**()

{

ui->textEdit\_2->clear();

ui->textEdit->clear();

ui->pushButton\_3->setDisabled(1);

ui->pushButton\_5->setDisabled(1);

QString x = "";

QString x1 = "";

int size = rand()%10 + 2;

if (flag == 1)

{

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

{

x += "[ ";

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

{

x = x +QString::number(rand()%2) + " ";

}

x += "]\n";

}

ui->textEdit\_2->setText(x);

}

else

{

vector<vector<QString>> a(size, vector<QString>(size, "0"));

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

{

for (int j=i ; j<size; j++)

{

a[i][j] = QString::number(rand()%2);

}

}

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

{

for (int j=0 ; j<=i; j++)

{

a[i][j] = a[j][i];

}

}

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

{

x += "[ ";

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

{

x = x + a[i][j] + " ";

}

x += "]\n";

}

ui->textEdit\_2->setText(x);

}

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

{

x1 = x1 + QString::number(i+1) + ": " + QString::number(i+1) + "\n";

}

ui->textEdit\_3->setText(x1);

}

void MainWindow::**updateMatrix**(const vector<vector<MyNode> > &a)

{

ui->textEdit\_2->clear();

QString x = "";

int size = a[0].size();

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

{

x += "[ ";

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

{

x = x + QString::number(a[i][j].value) + " ";

}

x +="]\n";

}

ui->textEdit\_2->setText(x);

}

void **changeMatrix**(vector<vector<MyNode>>& a, const QString& str, int change)

{

QStringList list = str.split(' ');

int ii, jj;

for (unsigned int i=0; i< a[0].size(); i++)

{

if (a[0][i].name == list[0])

ii = i;

if (a[0][i].name == list[1])

jj = i;

}

a[ii][jj].value = change;

if (flag == 0)

{

a[jj][ii].value = change;

}

}

// Connect 2 Nodes

void MainWindow::**on\_pushButton\_3\_clicked**()

{

vector<vector<MyNode>> a = textEditToVector();

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

changeMatrix(a, str, 1);

GraphWidget \*widget = new GraphWidget(a, ui->textEdit, flag, this);

ui->graphicsView->setScene(widget->scene());

ui->textEdit->clear();

updateMatrix(a);

}

// Connect All Nodes

void MainWindow::**on\_pushButton\_4\_clicked**()

{

QPointF p(0.0,0.0);

vector<vector<MyNode>> a = textEditToVector();

vector<vector<MyNode>> b(a[0].size(), vector<MyNode>(a[0].size(), {"", 1, p}));

updateMatrix(b);

on\_pushButton\_clicked();

}

// Remove 1 Edge

void MainWindow::**on\_pushButton\_5\_clicked**()

{

vector<vector<MyNode>> a = textEditToVector();

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

changeMatrix(a, str, 0);

GraphWidget \*widget = new GraphWidget(a, ui->textEdit, flag, this);

ui->graphicsView->setScene(widget->scene());

ui->textEdit->clear();

updateMatrix(a);

}

// Remove All Edges

void MainWindow::**on\_pushButton\_6\_clicked**()

{

vector<vector<MyNode>> a = textEditToVector();

vector<vector<MyNode>> b(a[0].size(), vector<MyNode>(a[0].size()));

updateMatrix(b);

on\_pushButton\_clicked();

}

void MainWindow::**on\_action\_triggered**()

{

QString fileName = QFileDialog::getOpenFileName(this, tr("Открыть файл"), QString(),

tr("Text Files (\*.txt)"));

ui->textEdit\_2->clear();

ui->textEdit\_3->clear();

if (!fileName.isEmpty())

{

QFile file(fileName);

if (!file.*open*(QIODevice::ReadOnly))

{

QMessageBox::critical(this, tr("Ошибка"), tr("Не могу открыть данный файл"));

return;

}

QTextStream in(&file);

QString x;

while (1)

{

x = in.readLine();

if (x != "#############")

ui->textEdit\_2->setText(ui->textEdit\_2->toPlainText() + x + "\n");

else

break;

}

while (1)

{

x = in.readLine();

if (x != "#############")

ui->textEdit\_3->setText(ui->textEdit\_3->toPlainText() + x + "\n");

else

break;

}

file.*close*();

}

}

void MainWindow::**on\_action\_2\_triggered**()

{

QString fileName = QFileDialog::getSaveFileName(this, tr("Сохранить файл"), QString(),

tr("Text Files (\*.txt)"));

if (!fileName.isEmpty())

{

QFile file(fileName);

if (!file.*open*(QIODevice::WriteOnly))

{

QMessageBox::critical(this, tr("Ошибка"), tr("Не могу открыть данный файл"));

return;

} else

{

QTextStream stream(&file);

stream << ui->textEdit\_2->toPlainText();

stream << "\n#############\n";

stream << ui->textEdit\_3->toPlainText();

stream << "\n#############\n";

stream.flush();

file.*close*();

}

}

}

void MainWindow::**on\_action\_4\_triggered**()

{

QMessageBox::about(this, tr("Об авторе"), tr("Лабораторная работа №2\n\nВыполнил: Ковынев М.В.\nГруппа: 6304"));

return;

}

void MainWindow::**on\_action\_3\_triggered**()

{

QMessageBox::about(this, tr("О программе"),

tr("Данная программа рисует ориентированный граф. "

"\n\Возможности:\n1. Сохранение графа в файл\n2."

" Возможность создание и удаления ребер по двойному "

"клику вершин\n3. Замена имени вершины графа на собственную\n4. "

"Поиск компонент двусвязности"));

return;

}

void MainWindow::**on\_pushButton\_7\_clicked**()

{

struct XY

{

int x;

int y;

};

ui->textEdit\_4->clear();

vector<vector<MyNode>> a = textEditToVector();

int size = 0;

if (a[0].size() != 0)

{

for (unsigned int i=0; i < a[0].size(); i++)

{

for (int j=0; j < i; j++)

if (a[i][j].value == 1)

size++;

}

vector<XY> b(size);

unsigned long long int k= 0;

for (unsigned int i=0; i < a[0].size(); i++)

{

for (unsigned int j=0; j < i; j++)

if (a[i][j].value == 1)

{

b[k].x = i;

b[k].y = j;

k++;

}

}

QString x= "";

int counter = 0;

k=powf(2,size);

for (unsigned int i=0; i<k; i++)

{

map <int, int> tt;

counter = 0;

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

{

if (i & (1<<j))

{

counter++;

tt[b[j].y] = 1;

tt[b[j].x] = 1;

}

}

if (counter != 0 )

{

int y =0;

for (auto& t: tt)

{

t.second = y;

y++;

}

Graph g1(tt.size());

x = x+ "Количество вершин: " + QString::number(counter + 1)+ "\nСписок ребер:\n{";

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

{

if (i & (1<<j))

{

x = x + "(" + a[b[j].x][b[j].x].name + ", " +a[b[j].y][b[j].y].name + ") ";

g1.addEdge(tt[b[j].x] , tt[b[j].y]);

}

}

if ( g1.isBC())

{

x += "}\nПодграф - компонента двусвязности\n";

}

else

{

x += "}\nПодграф - НЕ компонента двусвязности\n";

}

x+="######################################\n\n";

}

}

ui->textEdit\_4->setText(x);

}

}

void MainWindow::**on\_radioButton\_clicked**()

{

if (flag == 1)

{

flag = 0;

ui->pushButton\_7->setDisabled(0);

}

else

{

ui->pushButton\_7->setDisabled(1);

flag = 1;

}

}

# **ПРИЛОЖЕНИЕ К.**

# **ИХОДНЫЙ КОД NODE.H**

#ifndef NODE\_H

#define NODE\_H

#include <QGraphicsItem>

#include <QList>

#include <QTextEdit>

class Edge;

class GraphWidget;

class QGraphicsSceneMouseEvent;

class Node : public QGraphicsItem

{

public:

Node(QString, QTextEdit\* , QPointF, GraphWidget \*graphWidget);

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

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

enum { Type = UserType + 1 };

int ***type***() const override { return Type; }

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

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

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

QString maintext;

QTextEdit\* textEdit;

protected:

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

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

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

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

private:

QList<Edge \*> edgeList;

GraphWidget \*graph;

QPointF position;

};

#endif // NODE\_H

# **ПРИЛОЖЕНИЕ Л.**

# **ИХОДНЫЙ КОД NODE.CPP**

#include "edge.h"

#include "node.h"

#include "graphwidget.h"

#include <QGraphicsScene>

#include <QGraphicsSceneMouseEvent>

#include <QPainter>

#include <QStyleOption>

#include <fstream>

#include <QDebug>

using namespace std;

Node::**Node**(QString text, QTextEdit\* textEdit1, QPointF pos, GraphWidget \*graphWidget)

: graph(graphWidget)

{

setFlag(ItemIsMovable);

setFlag(ItemSendsGeometryChanges);

setCacheMode(DeviceCoordinateCache);

setZValue(-1);

maintext = text;

textEdit = textEdit1;

position = pos;

}

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

{

edgeList << edge;

edge->adjust();

}

QList<Edge \*> Node::**edges**() const

{

return edgeList;

}

QRectF Node::***boundingRect***() const

{

qreal adjust = 2;

return QRectF( -20 - adjust, -20 - adjust, 230 + adjust, 230 + adjust);

}

QPainterPath Node::***shape***() const

{

QPainterPath path;

path.addEllipse(-20, -20, 40, 40);

return path;

}

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

{

QRadialGradient gradient(-3, -3, 10);

if (option->state & QStyle::State\_Sunken)

gradient.setColorAt(0, QColor(Qt::darkYellow).light(120));

else

gradient.setColorAt(0, Qt::yellow);

painter->setBrush(gradient);

painter->setPen(QPen(Qt::black, 0));

painter->drawEllipse(-20, -20, 40, 40);

QRectF textRect(-5,-7, 1000, 1000);

painter->drawText(textRect, maintext);

}

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

{

switch (change)

{

case ItemPositionHasChanged:

foreach (Edge \*edge, edgeList)

edge->adjust();

graph->itemMoved();

break;

default:

break;

};

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

}

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

{

update();

QGraphicsItem::*mousePressEvent*(event);

}

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

{

update();

QGraphicsItem::*mouseReleaseEvent*(event);

}

void Node::***mouseDoubleClickEvent***(QGraphicsSceneMouseEvent \*event)

{

textEdit->setText(textEdit->toPlainText() + maintext + " ");

update();

QGraphicsItem::*mouseDoubleClickEvent*(event);

};