**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 «Построение и Анализ Алгоритмов»**

Тема: Максимальный поток в сети

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

Санкт-Петербург

2018

# **Цель работы**

Реализовать алгоритм нахождения максимального потока в сети.

# **Ход работы**

1. Реализуем функцию, которая обходит по заданному графу на ходит максимальный поток в сети.

Приведем описание алгоритма:

while(bfs(source,stock) && processing) // Пока существует путь

{

int delta = 10000000;

for(int u = stock; pred[u] >= 0 && processing; u = pred[u])

// Найти минимальный поток в пути

delta=qMin(delta,(capacity[pred[u]][u]-flow[pred[u]][u]));

for(int u = stock; pred[u] >= 0 && processing; u=pred[u])

// По алгоритму Форда-Фалкерсона

{

Node \* a = (\*getMscene()->nodesList())[pred[u]];

Node \* b = (\*getMscene()->nodesList())[u];

Edge \* e = getMscene()->findEdge(a,b);

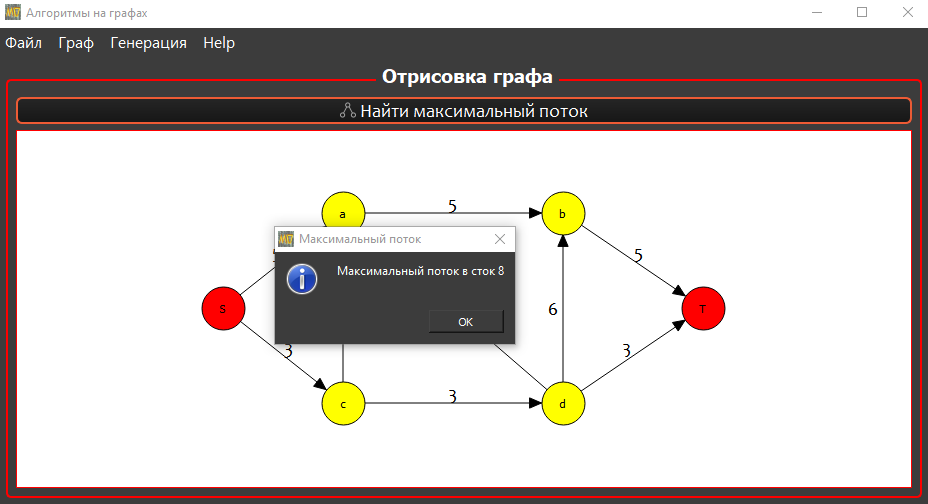
flow[pred[u]][u] += delta;

flow[u][pred[u]] -= delta;

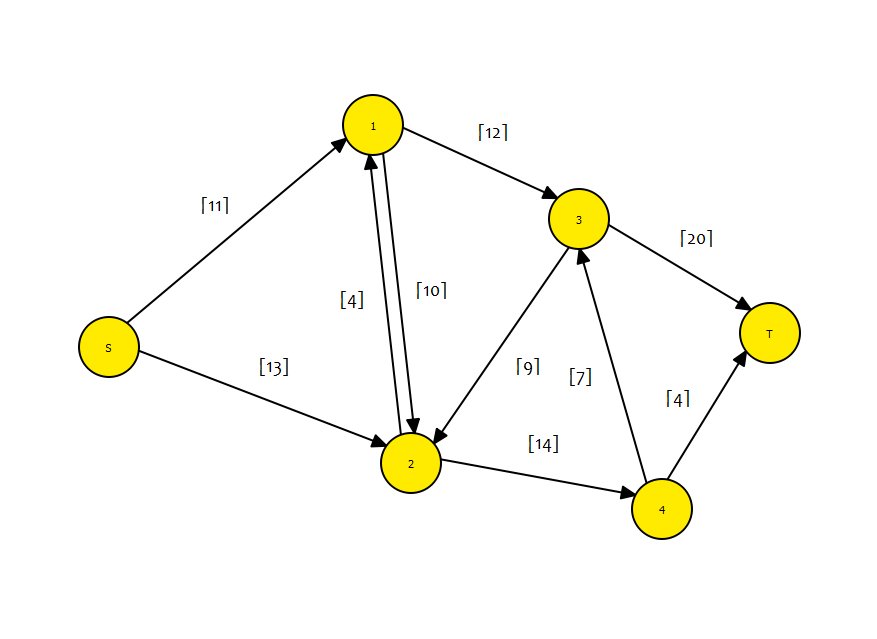
maxflow += delta; // Повышаем максимальный поток

}

return maxflow;



**Рисунок 1. Интерфейс программы.**

1. Проиллюстрируем работу алгоритма на примере графа, представленного на рис. 2.

**Рисунок 2. Ориентированный граф**

* 1. Исходная матрица пропускных способностей графа имеет вид (*S=0, T=5*):

[0 11 13 0 0 0]

[0 0 10 12 0 0]

[0 4 0 0 14 0]

[0 0 9 0 0 20]

[0 0 0 7 0 4]

[0 0 0 0 0 0]

* 1. Выполним поиск в глубину для данного графа. Результат: путь *T – 3 – 1 – S*. Потока в графе пока нет, следовательно, он равен 0. Найдем минимальную пропускную способность среди пар ребер *T – 3, 3 – 1, 1 – S*. Минимум среди 20, 12, 11 – это 11 (ребро *1 – S*). Значит ставим значения потока через данные ребра *T – 3, 3 – 1, 1 – S* равным11.
  2. Снова выполняем поиск. Результат: *T – 4 – 2 – S.* Потока через данные ребра тоже нет*.* Минимальный поток среди 4, 14, 13 – 4. Обновляем значения потока.
  3. Результат поиска в глубину: *T – 3 – 1 – 2 – S*. Поток через данные ребра равен 11, кроме ребра 2 – 1 (он равен 0). Минимальная разность между пропускной способностью ребер и потоком равна 1 – для ребра 3 – 1 (12 – 11 = 1). Добавляем 1 к каждому значению потока ребер *T – 3 – 1 – 2 – S.*
  4. Результат поиска в глубину: *T – 3 – 4 – 2 – S*. Пропускные способности для ребер *T – 3, 3 – 4 , 4 – 2, 2 – S* равны 12, 7, 14, 13 соответственно. Поток для данных ребер – 12, 0, 4, 5. Минимальная разность пропускной способности и потока равна 7 (7 *–* 0 = 7 ). Добавляем 7 к каждому значению потока ребер
  5. Алгоритм закончил свою работу. Путей больше нет. Максимальный поток в сток равен 23 (19 + 4 = 23)

[ 0 11 12 0 0 0 ]

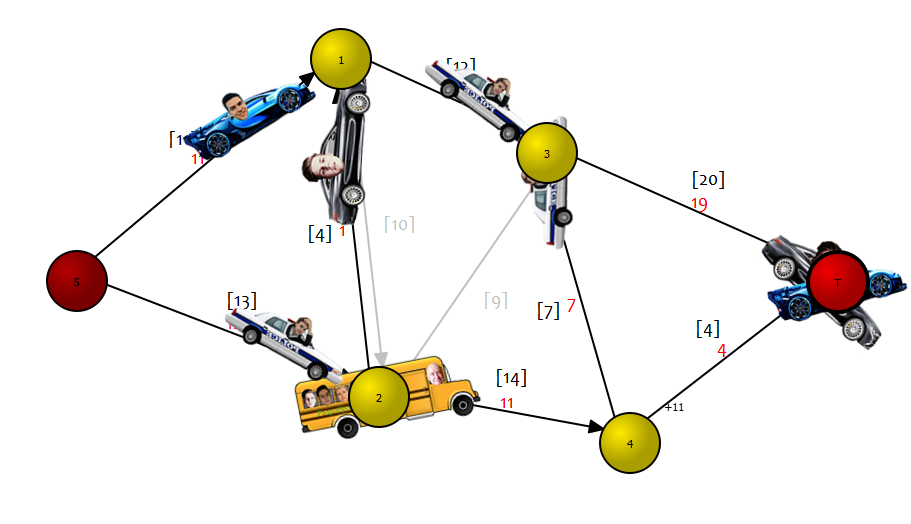
[-11 0 -1 12 0 0 ]

[-12 1 0 0 11 0 ]

[ 0 -12 0 0 -7 19 ]

[ 0 0 -11 7 0 4 ]

[ 0 0 0 -19 -4 0 ]

 **Рисунок 3. Результат поиска максимального потока.**

1. Реализуем анимацию передвижения транспортных средств по ребрам, у которых присутствует поток. Выбирается произвольная картинка из ресурсов проекта, запускается таймер анимации, который зависит от длины ребра.

void Edge::**animate**()

{

if(sourcePoint == destPoint)

return;

QPixmap p(":/cars/" +QString::number(qrand() % 8 + 1) +".png");

it->setPixmap(p); //установить изображение на объект

it->setOffset(-p.size().width()/2.0,-p.size().height()/2.0);

it->setZValue(0.5);

it->setPos(sourceNode()->scenePos());

it->setTransformationMode(Qt::SmoothTransformation);

posAnim->setItem(it);

updateAnimation();

scene->addItem(it);

timer->start();

}

Реализуем функцию обновления анимации. При каждом «тике» таймера необходимо обновлять расположение картинки. Для этого сначала рассчитаем траекторию движения транспортного средства, потом в зависимости от времени таймера будет ставить изображение на соответствующую точку линии движения.

void Edge::**updateAnimation**()

{

if(sourcePoint == destPoint)

return;

QLineF line(sourcePoint, destPoint); //линия соединения вершин

double angle = ::acos(line.dx() / line.length()); //угол наклона основной линии соединения

if (line.dy() >= 0)

angle = TwoPi - angle;

//средняя точка основной линии со смещением

QPointF c = 0.5\*line.p1() + 0.5\*line.p2() +QPointF(-sin(angle)\*40,cos(Pi - angle)\*40);

QPainterPath a; //построение кривой через три точки

a.moveTo(sourcePoint);

a.quadTo(c,destPoint);

timer->setDuration(a.length()/0.1); //расчет времени по длине ребра

bool ce = true; //для правильного расчета поворота

for(int i = 0; i <= 10; i++)

{

qreal p = i/10.0;

qreal p\_pred =0;

if(i>0)

p\_pred =(i-1)/10.0;

if(a.angleAtPercent(p) > a.angleAtPercent(p\_pred))

ce = false;

//установка позиции в разный момент времени

if(i==0)

posAnim->setPosAt(p,sourceNode()->scenePos());

else if(i==10)

posAnim->setPosAt(p,destNode()->scenePos());

else

posAnim->setPosAt(p,a.pointAtPercent(p));

//установка поворота в разные моменты времени

posAnim->setRotationAt(p, ce ? -a.angleAtPercent(p) : 360 - a.angleAtPercent(p));

}

}

# **Вывод**

В ходе выполнения данной лабораторной работы были изучены основные понятия и приёмы применения алгоритма нахождения максимального потока в сети.

# **ПРИЛОЖЕНИЕ А.**

# **ИХОДНЫЙ КОД CHOSE.H**

#ifndef CHOSE\_H

#define CHOSE\_H

#include <QDialog>

namespace Ui {

class chose;

}

class chose : public QDialog

{

Q\_OBJECT

public:

explicit chose(QWidget \*parent = 0);

~***chose***();

QString\* start\_name;

QString\* finish\_name;

private slots:

void **on\_buttonBox\_accepted**();

void **on\_buttonBox\_rejected**();

private:

Ui::chose \*ui;

};

#endif // CHOSE\_H

# **ПРИЛОЖЕНИЕ Б.**

# **ИХОДНЫЙ EGDE.H**

#ifndef EDGE\_H

#define EDGE\_H

#include <QGraphicsItem>

class Node;

class MScene;

//Класс для описания рёбер графа

class Edge : public QGraphicsItem

{

friend class Node;

friend class MScene;

public:

Edge(Node \*sourceNode, Node \*destNode, MScene \*sc, int val = 1);

~***Edge***(){qDebug() << "edge deleted";}

void **setColor**(QColor col); //установить цвет

void **removeEdge**(); //удалить ребро

void **setValue**(int n); //установить вес

Node \***sourceNode**() const; //получить вершину-родитель

Node \***destNode**() const; //получить вершину-ребенка

int **getValue**() const; //получить вес ребра

int **get\_flow\_value**();

int ***type***() const Q\_DECL\_OVERRIDE;

enum { Type = UserType + 2 }; //Необходимо для определения типа графического объекта

protected:

QRectF ***boundingRect***() const Q\_DECL\_OVERRIDE;

void ***paint***(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget) Q\_DECL\_OVERRIDE;

QPainterPath ***shape***() const Q\_DECL\_OVERRIDE;

void ***contextMenuEvent***(QGraphicsSceneContextMenuEvent \*event)Q\_DECL\_OVERRIDE;

void ***mouseDoubleClickEvent***(QGraphicsSceneMouseEvent \*event)Q\_DECL\_OVERRIDE;

private:

void **adjust**(); //Перерисовка линий при перетасквании Node

void **addMidNode**(QPointF pos); //Добавить промежуточную вершину

void **setOneArrowAtEdge**(bool ori);

Node \*source, \*dest; //Соединяемые вершины

QPointF sourcePoint; //Координаты концов линии

QPointF destPoint;

int flow\_value;

int value; //вес

bool have\_one\_arrow; //имеется ли у ребра две стрелки

QColor main\_color; //цвет

MScene\* scene; //сцена для отрисовки ребра

};

#endif // EDGE\_H

# **ПРИЛОЖЕНИЕ В.**

# **ИХОДНЫЙ КОД HELPBROWSER.H**

#ifndef HELPBROWSER\_H

#define HELPBROWSER\_H

#include <QtWidgets>

//класс для отображения текстового представления

//справки

//или окна настроек

class HelpBrowser : public QDialog{

Q\_OBJECT

public:

HelpBrowser(const QString& strPath,const QString& strFileName, QWidget\* pwgt = 0 );

//конструктор для окна справки

static bool **outDialog**(const QString& text, QString title);

//функция для вызова текстового представления

private:

HelpBrowser(const QString& text\_view);

//конструктор для окна текстового представления

int new\_diam; //параметры для окна настройки

int arr\_size;

bool oriented;

QGraphicsScene\* scene\_for\_set;

QString name;

void **readSettings**();

void **writeSettings**();

private slots:

void **set\_new\_diam**(int a){new\_diam = a;} //установка настроек

void **set\_oriented**(bool b){oriented = b;}

void **set\_new\_arr**(int a){arr\_size = a;}

void **ok\_changes**();

void **appl\_changes**();

// void appl\_changes(); //применить настройки

protected:

void ***closeEvent***(QCloseEvent \* event);

};

#endif // HELPBROWSER\_H

# **ПРИЛОЖЕНИЕ Г.**

# **ИХОДНЫЙ КОД MAINWINDOW.H**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QTextStream>

class Options;

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

friend class Options;

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~***MainWindow***();

private slots:

void **on\_action\_7\_triggered**();

void **on\_action\_8\_triggered**();

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

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

void **on\_action\_1\_triggered**();

void **on\_action\_13\_triggered**();

void **on\_action\_14\_triggered**();

void **on\_action\_15\_triggered**();

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

void **on\_action\_19\_triggered**();

void **on\_action\_20\_triggered**();

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

void **openFile**(QString temp);

QString **saveFile**();

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

void **on\_action\_21\_triggered**();

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

void **on\_action\_6\_triggered**();

private:

Ui::MainWindow \*ui;

bool processing;

int n,e;

int \*\*capacity; // Матрица пропускных способнотей

int \*\* flow; // Матрица потока

int \* color; // Цвета для вершин

int \* pred; // Массив пути

int head, tail; // Начало, Конец

int \* q; // Очере

void **enque**(int x);

int **deque**();

int **bfs**(int start,int end);

int **max\_flow**(int source,int stock);

};

#endif // MAINWINDOW\_H

# **ПРИЛОЖЕНИЕ Д.**

# **ИХОДНЫЙ КОД MGRAPHVIEW.H**

#ifndef MGRAPHVIEW\_H

#define MGRAPHVIEW\_H

#include <QtWidgets>

class MScene;

//Класс с виджетом

class MGraphView : public QGraphicsView

{

public:

MGraphView(QWidget \*parent = 0);

MScene\* getMscene(){return scene;}

~MGraphView();

protected:

void keyPressEvent(QKeyEvent \*event) Q\_DECL\_OVERRIDE;

void wheelEvent(QWheelEvent \*event) Q\_DECL\_OVERRIDE;

private:

void scaleView(qreal scaleFactor);

//увеличение масштаба

MScene\* scene;

};

#endif // MGRAPHVIEW\_H

# **ПРИЛОЖЕНИЕ Е.**

# **ИХОДНЫЙ КОД MSCENE.H**

#ifndef MSCENE\_H

#define MSCENE\_H

#include <QtWidgets>

class GraphWindow;

class MGraphView;

class Node;

class Edge;

class MainWindow;

class Options;

//Класс со сценой

class MScene : public QGraphicsScene

{

friend class GraphWindow;

friend class MGraphView;

friend class Node;

friend class Edge;

friend class MainWindow;

friend class Options;

public:

MScene(QObject\* par = 0);

~***MScene***();

//добавить вершину в заданной точке с заданным значением

void **addNode**(QPointF position, QString value = "");

//добавить ребро к заданным вершинам с заданным весом

void **addEdge**(Node\* source, Node\* dest, int val=1);

//установить диаметр для вершин

void **setDiametr**(int d);

//сделать граф ориентированным/ неориентированным

void **setOrientated**(bool b);

//установить размер стрелок

void **setArrowSize**(int d);

//включить/выключить отображение весов ребер

void **setShowEdge**(bool a);

int **diametr**() const;

int **arrSize**() const {return arrow\_size;}

bool **shouldShowValues**() const;

bool **isOriented**() const;

QList<Node\*> \* **nodesList**(); //список всех вершин

Node\* **findNode**(QString val); //найти вершину по значению

Edge\* **findEdge**(Node\* source,Node\* dest); //найти ребро по вершинам

MGraphView \* **getMGraph**(); //виджет

private:

void **setColorOfSelectedNodes**(); //Установка цвета выделенных вершин

//создание графа

void **createGraphWithText**(QList<QString> & nodes, QList<QStringList> & child\_of\_nodes);

QList<Node\*> all\_nodes; //все вершины на сцене

QList<QGraphicsItem\*> deleted; //все удаленные со сцены объекты (удаляются позже)

MGraphView \* parent; // виджет

Node\* first;

QGraphicsItem \* second;

bool show\_edge\_values; //показывать ли веса ребер

bool graph\_oriented; //ориентированный ли граф

int node\_diametr; //диаметр вершин

int arrow\_size; //размер стрелки

protected:

void ***mouseDoubleClickEvent***(QGraphicsSceneMouseEvent \* event) Q\_DECL\_OVERRIDE;

void ***mousePressEvent***(QGraphicsSceneMouseEvent \*event) Q\_DECL\_OVERRIDE;

private slots:

//соединить две вершины с помощью мыши

void **connectTwoNodes**();

public slots:

//удаление выделенных объектов

void **removeSelectedNodes**();

};

#endif // MSCENE\_H

# **ПРИЛОЖЕНИЕ Ж.**

# **ИХОДНЫЙ КОД MSCENE.H**

#ifndef NODE\_H

#define NODE\_H

#include <QGraphicsItem>

#include <QList>

class Edge;

class MGraphView;

class MScene;

//Класс для определения вершин графа

class Node : public QGraphicsItem

{

friend class Edge;

friend class MScene;

friend QString& **operator**<<(QString & str, const Node & node);

public:

Node(MScene \*graphWidget, QString val = "");

~***Node***(){qDebug() << "node deleted";}

void **setColor**(QColor col); //установить цвет

void **setValue**(QString val); //установить значение

void **removeNode**(); //удалить данную вершину

QString **get\_val**() const; //извлечение значения

QColor **get\_color**() const;

QList<Edge \*> \* **edges**(); //списки ребер и детей

QList<Node \*> \* **children**();

//соединены ли две вершины

static bool **is\_two\_nodes\_connected**(Node\* src, Node\* dst);

enum { Type = UserType + 1 }; //Тип "вершина"

int ***type***() const Q\_DECL\_OVERRIDE;

protected:

QRectF ***boundingRect***() const Q\_DECL\_OVERRIDE;

void ***paint***(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget) Q\_DECL\_OVERRIDE;

QPainterPath ***shape***() const Q\_DECL\_OVERRIDE;

QVariant ***itemChange***(GraphicsItemChange change, const QVariant &value) Q\_DECL\_OVERRIDE;

void ***contextMenuEvent***(QGraphicsSceneContextMenuEvent \*event)Q\_DECL\_OVERRIDE;

void ***mouseDoubleClickEvent***(QGraphicsSceneMouseEvent \*event)Q\_DECL\_OVERRIDE;

private:

void **addChild**(Node \* child); //Добавить ребенка к вершине

void **addEdge**(Edge \*edge); //Добавить ребро

QList<Edge \*> edgeList; //Список ребер

QList<Node \*> childrenList; //Смежные вершины

MScene \* scene;

QColor main\_color; //цвет

QString value; //значение

};

#endif // NODE\_H

# **ПРИЛОЖЕНИЕ И.**

# **ИХОДНЫЙ КОД MSCENE.H**

#ifndef OPTIONS\_H

#define OPTIONS\_H

#include <QDialog>

#include "mainwindow.h"

class MainWindow;

namespace Ui {

class Options;

}

class Options : public QDialog

{

friend class MainWindow;

Q\_OBJECT

public:

explicit Options(QWidget \*parent = 0);

~***Options***();

MainWindow\* ptr;

private slots:

void **on\_buttonBox\_accepted**();

void **on\_buttonBox\_rejected**();

private:

Ui::Options \*ui;

};

#endif // OPTIONS\_H

# **ПРИЛОЖЕНИЕ К.**

# **ИХОДНЫЙ КОД MSCENE.H**

#ifndef QCOMPRESSOR\_H

#define QCOMPRESSOR\_H

#include <zlib.h>

#include <QByteArray>

#define GZIP\_WINDOWS\_BIT 15 + 16

#define GZIP\_CHUNK\_SIZE 32 \* 1024

class QCompressor

{

public:

static bool **gzipCompress**(QByteArray input, QByteArray &output, int level = -1);

static bool **gzipDecompress**(QByteArray input, QByteArray &output);

};

#endif // QCOMPRESSOR\_H

l