МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Алгоритмы и структуры данных»
Тема: Рекурсия
Вариант 3

Студент гр. 8304 _____ Николаева М.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2019

Цель работы.

Ознакомиться с основными понятиями и приёмами рекурсивного программирования, получить навыки программирования рекурсивных процедур и функций на языке программирования C++ в фреймворке Qt.

Постановка задачи.

1) проанализировать полученное задание, выделив рекурсивно определяемые информационные объекты и (или) действия;
2) разработать программу, использующую рекурсию;
3) сопоставить рекурсивное решение с итеративным решением задачи;
4) сделать вывод о целесообразности и эффективности рекурсивного решения данной задачи.

Вариант 3: Имеется n городов, пронумерованных от 1 до n. Некоторые пары городов соединены дорогами. Определить, можно ли попасть по этим дорогам из одного заданного города в другой заданный город. Входная информация о дорогах задаётся в виде последовательности пар чисел i и j ($i < j$ и $i, j \in 1..n$), указывающих, что i-й и j-й города соединены дорогами.

Описание алгоритма.

Для вычисления значения выражения методом рекурсии необходимо запустить обход в глубину от вершины, из которой требуется найти путь. От вершины, на которой мы находимся на данном этапе рекурсии требуется проверить все связи, при этом вызов рекурсии происходит только в случае, если вершина не посещена. Рекурсия завершается при обходе всех возможных связей. В результате требуется посмотреть, посещена ли вершина, являющаяся конечным пунктом.

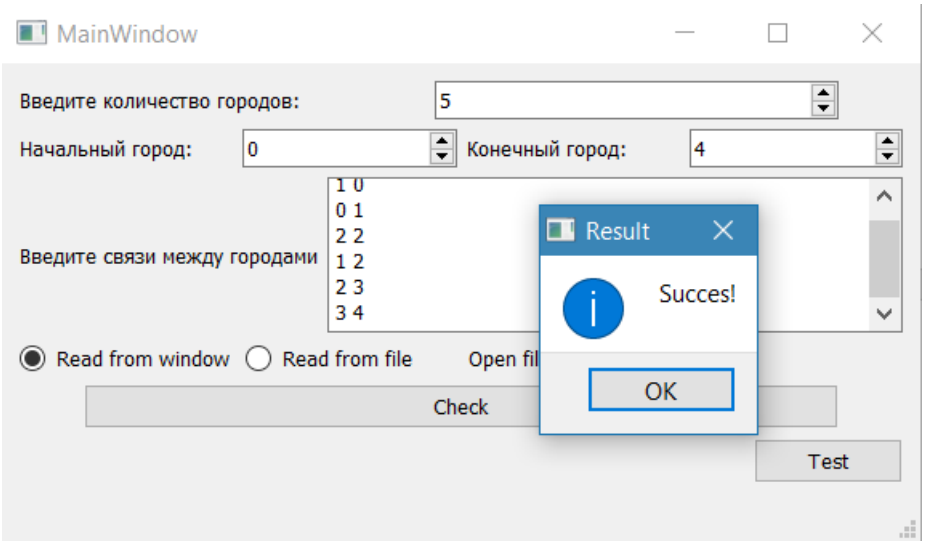
Спецификация программы.

Программа предназначена для поиска пути между двумя городами методом рекурсии.

Программа написана на языке C++. Входными данными являются числа: количество городов, начальный город, конечный город и связи между городами. Выходными данными являются промежуточные значения проверки

(записываются в qDebug()) и конечный результат (можно ли попасть из начального горда в конечный или нет). Данные выводятся в QMessageBox.

Рисунок 1 – Результат работы программы



Тестирование.

Рисунок 2 – Тестирование программы из папки Tests

```
Тестовые данные из файла: "12.txt"
The number of cities: 6
List of existing roads:
 Road from 0 to 1
 Road from 2 to 3
 Road from 3 to 5
Paving the route from 4 to 5
 We are currently in the city  4
 End of recursion
Sorry impossible to get directions
End PavingTheRoute

Тестовые данные из файла: "Novy_textovy_dokument_3.txt"
The number of cities: 5
List of existing roads:
 Road from 0 to 1
 Road from 1 to 2
 Road from 3 to 4
 Road from 2 to 3
Paving the route from 1 to 4
 We are currently in the city  1
   We are currently in the city  0
   End of recursion
   We are currently in the city  2
     We are currently in the city  3
       We are currently in the city  4
       End of recursion
     End of recursion
   End of recursion
 End of recursion
Success
End PavingTheRoute

Тестовые данные из файла: "test.txt"
The number of cities: 7
List of existing roads:
 Road from 0 to 2
 Road from 2 to 3
 Road from 3 to 4
 Road from 1 to 0
 Road from 5 to 6
Paving the route from 2 to 3
 We are currently in the city  2
   We are currently in the city  0
     We are currently in the city  1
     End of recursion
   End of recursion
   We are currently in the city  3
     We are currently in the city  4
     End of recursion
   End of recursion
 End of recursion
Success
End PavingTheRoute
```

Анализ алгоритма.

Для экономии памяти векторы передаются по ссылке, что позволяет избежать копирования. Алгоритм работает за линейное время от количества городов + количество путей. Недостаток рекурсивного алгоритма – ограниченный стек вызовов функций, что в свою очередь накладывает ограничение на размер входной строки, а также затраты производительности на вызов функций.

Описание функций и СД

Функция, реализующая алгоритм проверки возможности добраться из начального города в конечный.

```
static void PavingTheRoute(size_t i, std::vector<std::vector<size_t>>& roads, std::vector<bool>& visited, size_t depth = 0);
```

Принимает на вход номер текущего города, вектор путей, вектор посещенных городов, и глубину рекурсии для отладки.

После окончания работы алгоритма проверяется, был ли посещен конечный город.

Выводы.

В ходе работы были изучены рекурсивные методы решения задач, в частности DFS. Создан алгоритм, основанный на данном методе, для поиска путей между городами.

Приложение А.

mainwondow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QDebug>
#include <vector>
#include <string>
#include <QStringList>
#include <QMessageBox>
#include <QDir>
#include <QTextStream>
#include <QFile>
#include <QFileDialog>
```

```
namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void on_chek_pushButton_clicked();

    //Функция вычисления пути
    static void PavingTheRoute(size_t i, std::vector<std::vector<size_t>>& roads, std::vector<bool>&
visited, size_t depth = 0);

    void on_readFromFile_radioButton_clicked();

    void on_readFromWindow_radioButton_clicked();

    void on_test_pushButton_clicked();

private:
    Ui::MainWindow *ui;
    QDir* dir;
    QTextStream* in;
    QFile* file;
};

#endif // MAINWINDOW_H
```

main.cpp

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    dir = new QDir;
    in = new QTextStream;
    file = new QFile;

    ui->numberOfCity_spinBox->setRange(0, 1000);
    ui->startCity_spinBox->setRange(0, 1000);
    ui->endCity_spinBox->setRange(0, 1000);
    ui->readFromWindow_radioButton->setChecked(true);
}


MainWindow::~MainWindow()
{
    delete ui;
    delete file;
    delete dir;
    delete in;
}
```

```
void MainWindow::on_chek_pushButton_clicked()
{
    qDebug() << "Start PavingTheRoute";

    QStringList arrayStr;
    QString text;

    size_t startCity = 0;
    size_t endCity= 0;

    size_t numberOfCityes = 0;

    std::vector<std::vector <size_t>> roads;
    std::vector<bool> visited;

    size_t whereFrom = 0;
    size_t whereTo = 0;

    if (ui->readFromWindow_radioButton->isChecked()) {
        numberOfCityes = static_cast <size_t> (ui->numberOfCity_spinBox->value());

        if (numberOfCityes == 0) {
            qDebug() <<"Error, incorrect data";
            QMessageBox::critical(this, "Error", "Incorrect data!");
            return;
        } else {
            qDebug() <<"The number of cities:" << numberOfCityes;
        }

        roads.resize(numberOfCityes);
        visited.resize(numberOfCityes, false);

        startCity = static_cast <size_t> (ui->startCity_spinBox->value());
        endCity = static_cast <size_t> (ui->endCity_spinBox->value());

        text = ui->textEdit->toPlainText();
    }
    else if (ui->readFromFile_radioButton->isChecked()) {
        QString tmpStr = "";
        tmpStr = in->readLine();

        QStringList tmpList = tmpStr.split(" ");

        bool ok_0 = true;
        bool ok_1 = true;
        bool ok_2 = true;
        numberOfCityes = tmpList[0].toULongLong(&ok_2);
        startCity = tmpList[1].toULongLong(&ok_0);
        endCity = tmpList[2].toULongLong(&ok_1);

        if (ok_0 == false || ok_1 == false || ok_2 == false || numberOfCityes == 0
                || startCity >= numberOfCityes || endCity >= numberOfCityes ) {
            qDebug() <<"Error, incorrect data";
            QMessageBox::critical(this, "Error", "Incorrect data!");
            return;
        }

        if (numberOfCityes == 0) {
            qDebug() <<"Error, incorrect data";
            QMessageBox::critical(this, "Error", "Incorrect data!");
            return;
        } else {
            qDebug() <<"The number of cities:" << numberOfCityes;
        }

        roads.resize(numberOfCityes);
        visited.resize(numberOfCityes, false);

        text = in->readAll();
    }

    qDebug() << "List of existing roads:";

    arrayStr = text.split('\n');
    for (auto elem : arrayStr) {
        QStringList twoTown = elem.split(" ");

        if (twoTown.size() != 2) {
```

```
            qDebug() <<"Error, incorrect data";
            QMessageBox::critical(this, "Error", "Incorrect data!");
            return;
        }

        bool ok_0 = true;
        bool ok_1 = true;
        whereFrom = twoTown[0].toULongLong(&ok_0);
        whereTo = twoTown[1].toULongLong(&ok_1);

        if (ok_0 == false || ok_1 == false || whereFrom >= numberOfCityes || whereTo >=
numberOfCityes) {
            qDebug() <<"Error, incorrect data";
            QMessageBox::critical(this, "Error", "Incorrect data!");
            return;
        } else {
            qDebug() << " Road from" << whereFrom << "to" << whereTo;
        }

        roads[whereFrom].push_back(whereTo);
        roads[whereTo].push_back(whereFrom);
    }

    if (startCity >= numberOfCityes || endCity >= numberOfCityes){
        qDebug() <<"Error, incorrect data";
        QMessageBox::critical(this, "Error", "Incorrect data!");
        return;
    }

    qDebug() << "Paving the route from" << startCity << "to" << endCity;
    PavingTheRoute(startCity, roads, visited);

    if (visited[endCity]) {
        qDebug() << "Success";
        QMessageBox::information(this, "Result", "Succes!");
    }
    else {
        qDebug() << "Sorry impossible to get directions";
        QMessageBox::warning(this, "Result", "No way");
    }

    qDebug() << "End PavingTheRoute";

    ui->readFromWindow_radioButton->setChecked(true);
    emit on_readFromWindow_radioButton_clicked();
}


void MainWindow::PavingTheRoute(size_t i, std::vector<std::vector<size_t>>& roads,
std::vector<bool>& visited, size_t depth)
{
    visited[i] = true;

    std::string dbgStr = "";
    for (size_t i = 0; i < depth; ++i){
        dbgStr += "  ";
    }

    qDebug() << dbgStr.c_str() << "We are currently in the city " << i;

    for (size_t j = 0; j < roads[i].size(); ++j){
        if (visited[roads[i][j]] == false)
            PavingTheRoute(roads[i][j], roads, visited, depth + 1);
    }

    qDebug() << dbgStr.c_str() << "End of recursion";
}


void MainWindow::on_readFromFile_radioButton_clicked()
{
    file->close();

    ui->textLabel_0->setEnabled(false);
    ui->textLabel_1->setEnabled(false);
    ui->textLabel_2->setEnabled(false);
    ui->textLabel_3->setEnabled(false);
    ui->textEdit->setEnabled(false);
    ui->endCity_spinBox->setEnabled(false);
```

```
    ui->startCity_spinBox->setEnabled(false);
    ui->numberOfCity_spinBox->setEnabled(false);

    QString fileName = QFileDialog::getOpenFileName(this, "Open file", "", "*.txt");
    file->setFileName(fileName);
    file->open(QFile::ReadOnly | QFile::Text);
    in->setDevice(file);

    ui->openFile_textLabel->setText(fileName);
}


void MainWindow::on_readFromWindow_radioButton_clicked()
{
    file->close();

    ui->textLabel_0->setEnabled(true);
    ui->textLabel_1->setEnabled(true);
    ui->textLabel_2->setEnabled(true);
    ui->textLabel_3->setEnabled(true);
    ui->textEdit->setEnabled(true);
    ui->endCity_spinBox->setEnabled(true);
    ui->startCity_spinBox->setEnabled(true);
    ui->numberOfCity_spinBox->setEnabled(true);

    ui->openFile_textLabel->setText("none");
}

void MainWindow::on_test_pushButton_clicked()
{
    /*
     * Функция тестирования. Тестовые данные считываются из папки Tests
    */

    qDebug() << "Тестирование.";

    std::string expression = "";
    dir->cd(QApplication::applicationDirPath() + "/Tests");
    QStringList listFiles = dir->entryList(QStringList("*.txt"), QDir::Files);

    for (auto fileName : listFiles) {
        if (fileName == "." || fileName == "..")
            continue;

        qDebug();
        qDebug() << "Тестовые данные из файла:" << fileName;

        file->close();
        file->setFileName(dir->path() + "/" + fileName);
        file->open(QFile::ReadOnly | QFile::Text);
        in->setDevice(file);

        QStringList arrayStr;
        QString text;

        size_t startCity = 0;
        size_t endCity= 0;

        size_t numberOfCityes = 0;

        std::vector<std::vector <size_t>> roads;
        std::vector<bool> visited;

        size_t whereFrom = 0;
        size_t whereTo = 0;

        QString tmpStr = "";
        tmpStr = in->readLine();

        QStringList tmpList = tmpStr.split(" ");

        bool ok_0 = true;
        bool ok_1 = true;
        bool ok_2 = true;
        numberOfCityes = tmpList[0].toULongLong(&ok_2);
        startCity = tmpList[1].toULongLong(&ok_0);
        endCity = tmpList[2].toULongLong(&ok_1);

        if (ok_0 == false || ok_1 == false || ok_2 == false || numberOfCityes == 0
```

```
                || startCity >= numberOfCityes || endCity >= numberOfCityes ) {
            qDebug() <<"Error, incorrect data";
            QMessageBox::critical(this, "Error", "Incorrect data!");
            return;
        }

        if (numberOfCityes == 0) {
            qDebug() <<"Error, incorrect data";
            QMessageBox::critical(this, "Error", "Incorrect data!");
            return;
        } else {
            qDebug() <<"The number of cities:" << numberOfCityes;
        }

        roads.resize(numberOfCityes);
        visited.resize(numberOfCityes, false);

        text = in->readAll();

        qDebug() << "List of existing roads:";

        arrayStr = text.split('\n');
        for (auto elem : arrayStr) {
            QStringList twoTown = elem.split(" ");

            if (twoTown.size() != 2) {
                qDebug() <<"Error, incorrect data";
                QMessageBox::critical(this, "Error", "Incorrect data!");
                return;
            }

            bool ok_0 = true;
            bool ok_1 = true;
            whereFrom = twoTown[0].toULongLong(&ok_0);
            whereTo = twoTown[1].toULongLong(&ok_1);

            if (ok_0 == false || ok_1 == false || whereFrom >= numberOfCityes || whereTo >=
numberOfCityes) {
                qDebug() <<"Error, incorrect data";
                QMessageBox::critical(this, "Error", "Incorrect data!");
                return;
            } else {
                qDebug() << " Road from" << whereFrom << "to" << whereTo;
            }

            roads[whereFrom].push_back(whereTo);
            roads[whereTo].push_back(whereFrom);
        }

        if (startCity >= numberOfCityes || endCity >= numberOfCityes){
            qDebug() <<"Error, incorrect data";
            QMessageBox::critical(this, "Error", "Incorrect data!");
            return;
        }

        qDebug() << "Paving the route from" << startCity << "to" << endCity;
        PavingTheRoute(startCity, roads, visited);

        if (visited[endCity]) {
            qDebug() << "Success";
            QMessageBox::information(this, "Result", "Succes!");
        }
        else {
            qDebug() << "Sorry impossible to get directions";
            QMessageBox::warning(this, "Result", "No way");
        }

        qDebug() << "End PavingTheRoute";
    }
}
```

mainwindow.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
```

```
  <x>0</x>
  <y>0</y>
  <width>573</width>
  <height>298</height>
 </rect>
</property>
<property name="windowTitle">
 <string>MainWindow</string>
</property>
<widget class="QWidget" name="centralWidget">
 <layout class="QVBoxLayout" name="verticalLayout">
  <item>
   <layout class="QHBoxLayout" name="horizontalLayout_4">
    <item>
     <widget class="QLabel" name="textLabel_0">
      <property name="text">
       <string>Введите количество городов:</string>
      </property>
     </widget>
    </item>
    <item>
     <widget class="QSpinBox" name="numberOfCity_spinBox"/>
    </item>
    <item>
     <spacer name="horizontalSpacer_4">
      <property name="orientation">
       <enum>Qt::Horizontal</enum>
      </property>
      <property name="sizeType">
       <enum>QSizePolicy::Preferred</enum>
      </property>
      <property name="sizeHint" stdset="0">
       <size>
        <width>40</width>
        <height>20</height>
       </size>
      </property>
     </spacer>
    </item>
   </layout>
  </item>
  <item>
   <layout class="QHBoxLayout" name="horizontalLayout_5">
    <item>
     <widget class="QLabel" name="textLabel_2">
      <property name="text">
       <string>Начальный город:</string>
      </property>
     </widget>
    </item>
    <item>
     <widget class="QSpinBox" name="startCity_spinBox"/>
    </item>
    <item>
     <widget class="QLabel" name="textLabel_3">
      <property name="text">
       <string>Конечный город:</string>
      </property>
     </widget>
    </item>
    <item>
     <widget class="QSpinBox" name="endCity_spinBox"/>
    </item>
   </layout>
  </item>
  <item>
   <layout class="QHBoxLayout" name="horizontalLayout">
    <item>
     <widget class="QLabel" name="textLabel_1">
      <property name="text">
       <string>Введите связи между городами</string>
      </property>
     </widget>
    </item>
    <item>
     <widget class="QTextEdit" name="textEdit"/>
    </item>
   </layout>
  </item>
```

```
<item>
 <layout class="QHBoxLayout" name="horizontalLayout_6">
  <item>
   <widget class="QRadioButton" name="readFromWindow_radioButton">
    <property name="text">
     <string>Read from window</string>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QRadioButton" name="readFromFile_radioButton">
    <property name="text">
     <string>Read from file</string>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QLabel" name="textLabel_4">
    <property name="text">
     <string>Open file:</string>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QLabel" name="openFile_textLabel">
    <property name="text">
     <string>none</string>
    </property>
   </widget>
  </item>
 </layout>
</item>
<item>
 <layout class="QHBoxLayout" name="horizontalLayout_2">
  <item>
   <spacer name="horizontalSpacer">
    <property name="orientation">
     <enum>Qt::Horizontal</enum>
    </property>
    <property name="sizeType">
     <enum>QSizePolicy::Minimum</enum>
    </property>
    <property name="sizeHint" stdset="0">
     <size>
      <width>40</width>
      <height>20</height>
     </size>
    </property>
   </spacer>
  </item>
  <item>
   <widget class="QPushButton" name="chek_pushButton">
    <property name="text">
     <string>Check</string>
    </property>
   </widget>
  </item>
  <item>
   <spacer name="horizontalSpacer_2">
    <property name="orientation">
     <enum>Qt::Horizontal</enum>
    </property>
    <property name="sizeType">
     <enum>QSizePolicy::Minimum</enum>
    </property>
    <property name="sizeHint" stdset="0">
     <size>
      <width>40</width>
      <height>20</height>
     </size>
    </property>
   </spacer>
  </item>
 </layout>
</item>
<item>
 <layout class="QHBoxLayout" name="horizontalLayout_3">
  <item>
   <spacer name="horizontalSpacer_3">
```

```
        <property name="orientation">
         <enum>Qt::Horizontal</enum>
        </property>
        <property name="sizeHint" stdset="0">
         <size>
          <width>40</width>
          <height>20</height>
         </size>
        </property>
       </spacer>
      </item>
      <item>
       <widget class="QPushButton" name="test_pushButton">
        <property name="text">
         <string>Test</string>
        </property>
       </widget>
      </item>
     </layout>
    </item>
   </layout>
  </widget>
  <widget class="QMenuBar" name="menuBar">
   <property name="geometry">
    <rect>
     <x>0</x>
     <y>0</y>
     <width>573</width>
     <height>26</height>
    </rect>
   </property>
  </widget>
  <widget class="QStatusBar" name="statusBar"/>
 </widget>
 <layoutdefault spacing="6" margin="11"/>
 <resources/>
 <connections/>
</ui>
```