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»

Студентка гр. 8381 Бердникова А.А.

Преподаватель Жангиров Т.Р.

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.

Основные теоретические положения

Рекурсия — определение, описание, изображение какого-либо объекта или процесса внутри самого этого объекта или процесса, то есть ситуация, когда объект является частью самого себя. В программировании рекурсией называется процедура, когда функция вызывает саму себя.

Задание

Вариант 2

Задано конечное множество имен жителей некоторого города, причем для каждого из жителей перечислены имена его детей. Жители X и Y называются родственниками, если (а) либо X – ребенок Y, (б) либо Y – ребенок X, (в) либо существует некоторый Z, такой, что X является родственником Z, а Z является родственником Y. Перечислить все пары жителей города, которые являются родственниками.

Описание алгоритма

На вход алгоритму подаются строки с именами жителей. Затем происходит построчное чтение.

Затем происходит разделение каждой строки по пробелам, после чего создается очередной элемент модифицированного словаря, организованного из Qvector<String> и в этот словарь добавляются все элементы из строки. Первый элемент(имя) и есть родитель остальных. Также в дополнительный список записываются имена всех жителей, неизвестных ранее.

После завершения считывания для каждого имени из списка всех имен вызывается рекурсивная bool функция, отвечающая на вопрос, являются ли два жителя родственниками. В случае положительного ответа эта пара выводися на экран.

Тестирование программы

Результаты выполнения программы на некоторых тестовых данных приведены в табл 1.

Табл 1 - Тестирование программы

Входные данные	Результат работы программы
Mitya Alya Gricha Dasha Mitya Alya Kostya Kostya Ilya	Mitya & Alya Mitya & Gricha Mitya & Dasha Mitya & Kostya Mitya & Ilya Alya & Gricha Alya & Dasha Alya & Kostya Alya & Ilya Gricha & Dasha Gricha & Kostya Gricha & Ilya Dasha & Kostya Dasha & Ilya Kostya & Ilya
B D F D G R B Q B	B & D B & F B & G B & R B & Q D & F D & G D & R D & Q F & G F & R F & Q G & R G & Q R & Q
B F G	

Анализ алгоритма.

Алгоритм имеет сложность O(n^2), где n – число жителей. Тем не менее т.к. алгоритм использует рекуррентные соотношения, осложняется возможность точно вычислить сложность. Одним из недостатков алгоритмов является использование рекурсии, что ограничивает количество вложенных операций.

Вывод

В ходе выполнения лабораторной работы были усвоены методы использования рекурсии, а также написана программа, использующая рекурсию для выполнения поставленной задачи. Выявлено, что данный метод достаточно эффективен для выполнения поставленной задачи.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

family.cpp

```
#include "family.h"


void Family::addWord(QString &name){
  bool here = false;
  for(auto tmp: all_names){
     if (tmp == name) {here = true; break;}
  }
  if(!here) all_names.push_back(name);
}


void Family::split(QString& row){
  families.push_back(QVector<QString>());
  QStringList list_of_strings = row.split(' ');
  if (list_of_strings.length()==0) throw WRONG_STRING;
  for (auto str:list_of_strings){
     if(str != ""){
        families[number].push_back(str);
        addWord(str);
     }
  }
  number++;
}



void Family::input(QString in){
  split(in);

}

bool Family::is_parent(QString p1, QString p2){
  for (auto & familie : families) {
     if (p1==familie[0]){
        for (auto j = 1; j <familie.size() ; ++j) {
           if (p2 == familie[j]) return true;
        }
     }
  }
  return false;
```

```
}


bool Family::is_relative(const QString& p1, const QString& p2, int deep){
  if (p1==p2) return false;
  if (is_parent(p1,p2) || is_parent(p2,p1)) return true;
  if (deep>3) return false;
  for( auto Z : all_names)
  {
     if (is_relative(p1,Z,deep+1) && is_relative(Z,p2,deep+1)) return true;
  }
  return false;
}
```

about.cpp

```
#include "about.h"
#include "ui_about.h"

about::about(QWidget *parent) :
  QDialog(parent),
  ui(new Ui::about)
{
  ui->setupUi(this);
//    QPixmap myPixmap("qrc:/img/img/cover.jpg");
//    ui->YA->setPixmap( myPixmap );
}

about::~about()
{
  delete ui;
}
```

help.cpp

```
#include "help.h"
#include "ui_help.h"

help::help(QWidget *parent) :
  QDialog(parent),
  ui(new Ui::help)
{
  ui->setupUi(this);
}

help::~help()
```

```
{
  delete ui;
}
```

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);
  abt = new (about);
  hlp = new (help);
}

MainWindow::~MainWindow()
{
  delete ui;
}

void MainWindow::on_resultButtom_clicked()
{
  try {
        fml = new Family();
        QStringList list1 = ui->Input->toPlainText().split('\n');
        fml->COUNT=list1.length();
        if(can) {
           for (int i=0;i<fml->COUNT;i++) {
              if(list1[i].length()>0)
                 fml->input(list1[i]);
           }
```

```
        }
        QString out= "";
        for (auto i=0;i<fml->all_names.size();i++){
            for (auto j=i;j<fml->all_names.size();j++){
                QString X = fml->all_names[i];
                QString Y = fml->all_names[j];
                if (fml->is_relative(X,Y)) {
                    out += (X+" & "+Y+"\n");
                }
                ui->Output->setText(out);
            }
        }
    }
    catch (errors  e) {
        switch (e) {
            case WRONG_STRING:
            QMessageBox::critical(this, "Error", "Wrong string");
        }
    }
}

void MainWindow::on_Input_textChanged()
{
    can = true;
}




void MainWindow::on_actionOpen_file_triggered()
{
    QString fileName = QFileDialog().getOpenFileName();

    if (!fileName.isNull()){

        QFile file(fileName);

        if (file.open(file.ReadOnly)){
            QString data = file.readAll();
            ui->Input->setText(data);
        }
        else {
            QMessageBox::warning(this, "Error", "Error to open file");
        }
    }
```

```
}

void MainWindow::on_actionHelp_triggered()
{
  hlp->show();
}

void MainWindow::on_actionAbout_triggered()
{
  abt->show();
}

void MainWindow::on_actionExit_triggered()
{
  MainWindow::close();
}
```