Федеральное государственное бюджетное образовательное учреждение высшего образования «Саратовский государственный технический университет имени Гагарина Ю.А.»

Институт прикладных информационных технологий и коммуникаций

Кафедра Информационная безопасность автоматизированных систем

Направление подготовки 10.03.01 Информационная безопасность

Расчётно-графическая работа по дисциплине «Языки программирования»

«Тема»

Выполнил: студент 1 курса

учебной группы с-ИБС11

очной формы обучения

Одиноков А.А.

Проверил: ассистент каф. ИБС

Романчук С. П.

Саратов 2020

Содержание

Аннотация 3ст

Введение 4ст

Теоретическая часть 5ст

Практическая часть 8ст

Заключение 12ст

Приложение 13ст

Аннотация

Ремонт дорог

Дан файл, описывающий карту дорог в городе. Возможный формат файла описан ниже

(комментарии в реальном файле будут отсутствовать):

N // число перекрестков

X1 Y1 // координаты перекрестка

X2 Y2

...

XN YN

M // число дорог

N1 K1 // порядковые номера соединяемых перекрестков

...

NM KM // порядковые номера соединяемых перекрестков

Написать программу, которая для заданного файла рисует на экране карту (возможными

искривлениями дорог при отображении пренебречь). По выбору пользователя

принимается решение о закрытии дороги (или дорог) на ремонт. Программа должна

определять, можно ли закрыть дорогу. Дорогу закрывать нельзя, если при ее закрытии

нарушится связь между любыми двумя частями города.

Введение

Тема работы:

«Ремонт дорог»

Цель работы:

Написать программу, которая для заданного файла рисует на экране карту (возможными

искривлениями дорог при отображении пренебречь). По выбору пользователя

принимается решение о закрытии дороги (или дорог) на ремонт.

Задачи работы:

Программа должна определять, можно ли закрыть дорогу. Дорогу закрывать нельзя, если при ее закрытии нарушится связь между любыми двумя частями города.

Теоретическая часть

**C++** — компилируемый, статически типизированный язык программирования общего назначения.

Поддерживает такие парадигмы программирования как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование, обеспечивает модульность, раздельную компиляцию, обработку исключений, абстракцию данных, объявление типов (классов) объектов, виртуальные функции. Стандартная библиотека включает, в том числе, общеупотребительные контейнеры и алгоритмы. C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков. В сравнении с его предшественником — языком C, — наибольшее внимание уделено поддержке объектно-ориентированного и обобщённого программирования.

C++ широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. Область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также развлекательных приложений. Существует множество реализаций языка C++, как бесплатных, так и коммерческих и для различных платформ.

Типы:

Символьные: char, wchar\_t (char16\_t и char32\_t, в стандарте C++11).

Целочисленные знаковые: signed char, short int, int, long int (и long long int, в стандарте C++11).

Целочисленные беззнаковые: unsigned char, unsigned short int, unsigned int, unsigned long int(и unsigned long long int, в стандарте C++11).

С плавающей точкой: float, double, long double.

Логический: bool.

Операции сравнения возвращают тип bool. Выражения в скобках после if, while приводятся к типу bool.

Функции могут принимать аргументы по ссылке. Функции могут возвращать результат по ссылке. Cсылки сходны с указателями, со следующими особенностями: перед использованием ссылка должна быть инициализирована; ссылка всегда указывает на один и тот же адрес; в выражении ссылка обозначает непосредственно тот объект или ту функцию, на которую она указывает, обращение же к объекту или функции через указатель требует разыменование указателя.

Объектно-ориентированные особенности

C++ добавляет к C объектно-ориентированные возможности. Он вводит классы, которые обеспечивают три самых важных свойства ООП: инкапсуляцию, наследование и полиморфизм.

Методы класса — это функции, которые смогут применяться к экземплярам класса. Грубо говоря, метод — это функция объявленная внутри класса и предназначенная для работы с его объектами. Методы объявляются в теле класса. Описываться могут там же, но могут и за пределами класса (внутри класса в таком случае достаточно представить прототип метода, а за пределами класса определять метод поставив перед его именем — имя класса и оператор ::). Методы и поля входящие в состав класса называются членами класса. При этом методы часто называют функциями-членами класса.

Наследование

В C++ при наследовании одного класса от другого наследуется реализация класса, плюс класс-наследник может добавлять свои поля и функции или переопределять функции базового класса. Множественное наследование разрешено. Конструктор наследника вызывает конструкторы базовых классов, а затем конструкторы нестатических членов-данных, являющихся экземплярами классов. Деструктор работает в обратном порядке. Наследование бывает публичным, защищённым и закрытым.

Полиморфизм

Целью полиморфизма, применительно к объектно-ориентированному программированию, является использование одного имени для задания общих для класса действий. Выполнение каждого конкретного действия будет определяться типом данных. Преимуществом полиморфизма является то, что он помогает снижать сложность программ, разрешая использование того же интерфейса для задания единого класса действий. Выбор же конкретного действия, в зависимости от ситуации, возлагается на компилятор. Полиморфизм может применяться также и к операторам.

Стандартная библиотека

В языке программирования C++ термин Стандартная Библиотека означает коллекцию классов и функций, написанных на базовом языке. Стандартная Библиотека поддерживает несколько основных контейнеров, функций для работы с этими контейнерами, объектов-функции, основных типов строк и потоков (включая интерактивный и файловый ввод-вывод), поддержку некоторых языковых особенностей, и часто используемые функции для выполнения таких задач, как, например, нахождение квадратного корня числа. Стандартная Библиотека языка C++ также включает в себя спецификации стандарта ISO C90 стандартной библиотеки языка Си. Функциональные особенности Стандартной Библиотеки объявляются внутри пространства имен std.

Достоинства:

* Высокая совместимость с языком Си
* Вычислительная производительность
* Поддержка различных стилей программирования: структурное, объектно-ориентированное, обобщённое программирование, функциональное программирование, порождающее метапрограммирование.
* Автоматический вызов деструкторов объектов (в порядке обратном вызову конструкторов) упрощает и повышает надёжность управления памятью и другими ресурсами (открытыми файлами, сетевыми соединениями, т. п.).
* Перегрузка операторов
* Шаблоны (дают возможность построения обобщённых контейнеров и алгоритмов для разных типов данных)
* Возможность расширения языка для поддержки парадигм, которые не поддерживаются компиляторами напрямую
* Доступность. Для С++ существует огромное количество учебной литературы, переведённой на всевозможные языки

Недостатки:

* Плохо продуманный синтаксис сужает спектр применимости языка
* Язык не содержит многих важных возможностей
* Язык содержит опасные возможности
* Производительность труда программистов на языке оказывается неоправданно низка
* Громоздкость синтаксиса
* Тяжелое наследие
* Необходимость следить за памятью

**Qt Creator** (ранее известная под кодовым названием *Greenhouse*) — кроссплатформенная свободная IDE для разработки на С, С++ и QML. Разработана Trolltech *(Digia)* для работы с фреймворком Qt. Включает в себя графический интерфейс отладчика и визуальные средства разработки интерфейса как с использованием QtWidgets, так и QML. Поддерживаемые компиляторы: GCC, Clang, MinGW, MSVC, Linux ICC, GCCE, RVCT, WINSCW.

Практическая часть

В данной программе я использовал библиотеки

[#include](https://vk.com/im?sel=523604617&st=%23include) <QMainWindow>

Главное окно предоставляет структуру для создания пользовательского интерфейса приложения. Qt имеет класс QMainWindow и [связанные с ним классы](http://doc.crossplatform.ru/qt/4.7.x/mainwindow-classes.html) для управления главным окном. QMainWindow имеет собственный компоновщик, в который вы можете добавлять [QToolBar](http://doc.crossplatform.ru/qt/4.7.x/qtoolbar.html)'ы, [QDockWidget](http://doc.crossplatform.ru/qt/4.7.x/qdockwidget.html)'ы, [QMenuBar](http://doc.crossplatform.ru/qt/4.7.x/qmenubar.html), и [QStatusBar](http://doc.crossplatform.ru/qt/4.7.x/qstatusbar.html). Компоновщик имеет центральную область, которая может быть занята любым виджетом. Вы можете увидеть изображение компоновщика ниже.  
[#include](https://vk.com/im?sel=523604617&st=%23include) <QApplication>

Класс QApplication руководит управляющей логикой ГПИ и основными настройками. QApplication содержит основной цикл событий, в котором все события от оконной системы и других источников обрабатываются и координируются. Он также обрабатывает инициализацию и завершение приложения и обеспечивает управление сессиями.  
[#include](https://vk.com/im?sel=523604617&st=%23include) <exception>

Exception - это базовый класс, от которого наследуются исключения.  
[#include](https://vk.com/im?sel=523604617&st=%23include) <memory>

Заголовок определяет общие утилиты для управления динамической памятью  
[#include](https://vk.com/im?sel=523604617&st=%23include) <QList>  
Класс QList - это шаблонный класс, который предоставляет списки.  
[#include](https://vk.com/im?sel=523604617&st=%23include) <QPair>

Класс QPair - это шаблонный класс, в котором хранится пара элементов.  
[#include](https://vk.com/im?sel=523604617&st=%23include) <exception>  
[#include](https://vk.com/im?sel=523604617&st=%23include) <sstream>

Заголовок, предоставляющий классы потока строк  
[#include](https://vk.com/im?sel=523604617&st=%23include) <QPoint>

Точка задается координатами x и y, доступ к которым можно получить с помощью функций x () и y ().Объект QPoint также может быть использован в качестве вектора: сложение и вычитание определяются как для векторов (каждый компонент добавляется отдельно). Кроме того, класс QPoint предоставляет функцию manhattanLength (), которая дает недорогую аппроксимацию длины объекта QPoint, интерпретируемого как вектор. Наконец, объекты QPoint можно передавать в потоковом режиме, а также сравнивать.

class TextFileParser- Класс textFileParser является реализацией анализатора графов в текстовом файле.

class GraphOnAdjacencyMatrix- Класс TextFileParser является реализацией анализатора графов в текстовом файле.

class GraphOnAdjacencyMatrixBuilder- Класс GraphOnAdjacencyMatrixBuilder является компоновщиком для реализации графа, который хранит ребра в матрице несвязности.

class CanNotBuildAssociatedGraph- Класс CanNotBuildAssociatedGraph является исключением, которое улавливает ситуацию, когда граф после сборки не будет связан.

class IGraphBuilder- это интерфейс для построения графа.

virtual IGraphBuilder\* addVertex(int coordinateX, int coordinateY) = 0;

addVertex - это функция для добавления новой вершины в граф.

координата X - горизонтальная координата вершины.

координата Y - вертикальная координата й вершины.

virtual IGraphBuilder\* addVertex(int coordinateX, int coordinateY) = 0;

addEdge - это функция для добавления нового ребра на график.

firstVertex - это вершина, которая является началом ребра.

secondVertex - это вершина, которая является концом ребра.

throw(VertexNotFoundException) = 0;

функция для построения графа.

class IGraphComplete- Класс IGraphComplete является интерфейсом, который описывает график с полным желаемым функционалом.

class CanNotRemoveEdgeException- Класс CanNotRemoveEdgeException является исключением, которое улавливает ситуацию, когда удаление края порвет график.

class IGraphEdgeRemover- это интерфейс для удаления ребра из графа.

virtual void removeEdge(int firstVertex,int secondVertex)

removeEdge - функция для удаления края от графа.

firstVertex - это вершина, которая является началом ребра.

secondVertex - это вершина, которая является концом ребра.

typedef QList<QPair<int, int>> EdgeList;

это список из пар вершин.

class IGraphEdgesProvider- это интерфейс для предоставления всех ребер графа.

virtual EdgeList provideEdges() = 0;

это функция для получения всех ребер графа.

class CanNotParceGraphException- Класс CanNotParceGraphException является исключением, которое улавливает ситуацию, когда граф не может быть проанализирован.

class IGraphParser- это интерфейс для разбора графа из любого хранилища и сохранения его.

virtual ::std::shared\_ptr<IGraphComplete> provideGraph() throw (CanNotParceGraphException)= 0;

это функция, которая предоставляет проанализированный граф.

указатель на этот график.

virtual void saveGraph() = 0;  
saveGraph сохраняет обновленный график в хранилище.

class IGraphProvider- это интерфейс для предоставления информации о вершинах и ребрах графа.

class IGraphVertexAssociatedAdder- это интерфейс, обеспечивающий функционал для добавления вершин, которые будут связаны с графом.

virtual void addVertex(QPoint const& coordinates,int associatedVertex) throw(***VertexNotFoundException***) = 0; - addVertex - это функция для добавления новых связанных вершин.

Координаты - это координаты для новой вершины .

associatedVertex - это вершина, которая будет связана с новой вершиной.

virtual void addVertex(int coordinateX,int coordinateY,int associatedVertex) throw(VertexNotFoundException) = 0;

addVertex - это функция для добавления связанной вершины. Координата - горизонтальная координата новой вершины. Координата - вертикальная координата новой вершины. relatedVertex - это вершина, которая будет связана с новой вершиной.

typedef QList<QPair<int, int>> VertexList;

VertexList - список вершин. Каждая вершина имеет координаты x и y в качестве первого и второго элемента пары.

class IGraphVerticesProvider - это интерфейс для предоставления всех вершин графа.

virtual VertexList provideVertex() = 0;   
provideVertex - это функция для получения всех вершин графа. список вершин.

class VertexNotFoundException- Класс VertexNotFoundException является исключением, которое улавливает ситуацию, когда вершины, с которой вы хотите связать новую вершину, не существует.

class MyTextEdit- отрезает грани по ввведённому тексту

Заключение

Написана программа, которая для заданного файла рисует на экране карту (возможными

искривлениями дорог при отображении пренебречь). По выбору пользователя

принимается решение о закрытии дороги (или дорог) на ремонт.

Все поставленные задачи были выполнены.

В современном мире графы являются незаменимой вещью для программистов, так что эта программа будет полезной для программиста.

Приложение

#include <QDebug>

#include <QKeyEvent>

#include <QTextEdit>

#include <QGraphicsLineItem>

#include <QGraphicsScene>

#include <QGraphicsTextItem>

#include <QGraphicsView>

#include <QLabel>

#include <QVBoxLayout>

#include "mainwindow.h"

#include "MyTextEdit.h"

#include "file\_parser/TextFileParser.h"

#include "graph\_implemented/GraphOnAdjacencyMatrixBuilder.h"

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

TextFileParser\* t = new TextFileParser(

new GraphOnAdjacencyMatrixBuilder(),

":/res/graph"

);

try {

this->graph = t->*provideGraph*();

qDebug() << "vertices";

VertexList vl = graph->*provideVertex*();

for (QPair<int, int> vertex: vl) {

qDebug() << vertex.first << " " << vertex.second;

}

qDebug() << "edges";

EdgeList el = graph->*provideEdges*();

for (QPair<int, int> edge: el) {

qDebug() << edge.first << " " << edge.second;

}

/\*try {

graph->removeEdge(2, 3);

} catch (CanNotRemoveEdgeException e) {

qDebug() << "remove E "<< e.what();

} catch (VertexNotFoundException e) {

qDebug() << "remove E "<< e.what();

}

qDebug() << "remove without e";

qDebug() << "vertices";

VertexList vl2 = graph->provideVertex();

for (QPair<int, int> vertex: vl2) {

qDebug() << vertex.first << " " << vertex.second;

}

qDebug() << "edges";

EdgeList el2 = graph->provideEdges();

for (QPair<int, int> edge: el2) {

qDebug() << edge.first << " " << edge.second;

}\*/

} catch (CanNotParceGraphException e) {

qDebug() << e.*what*();

} catch (...) {

qDebug() << "anotherEXC";

}

qDebug() << "without exception";

this->scene = new QGraphicsScene(this);

scene->setSceneRect(0, 0, this->sceneWidth, this->sceneHeight);

QGraphicsView\* viewForScene = new QGraphicsView(this);

viewForScene->setScene(scene);

viewForScene->setMouseTracking(true);

viewForScene->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);

viewForScene->setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);

viewForScene->setRenderHint(QPainter::Antialiasing, false);

informationLabel = new QLabel(this);

informationLabel->setText("TEST");

MyTextEdit\* editText = new MyTextEdit();

QVBoxLayout\* rootLayout = new QVBoxLayout(this);

rootLayout->addWidget(informationLabel);

rootLayout->addWidget(editText);

rootLayout->addWidget(viewForScene);

connect(editText, SIGNAL(removeEdge(int, int)), this, SLOT(removeEdge(int, int)));

// VertexList vl = ::std::dynamic\_pointer\_cast<IGraphVerticesProvider, IGraphComplete>(graph).get()->provideVertex();

// = ( (IGraphVerticesProvider\*) graph.get())->provideVertex();

this->setLayout(rootLayout);

}

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

delete this->scene;

delete this->informationLabel;

}

void MainWindow::**printGraph**() {

VertexList vl = graph->*provideVertex*();

int maxX = 0, maxY = 0;

for (QPair<int, int> vertex: vl) {

if (vertex.first > maxX) maxX = vertex.first;

if (vertex.second > maxY) maxY = vertex.second;

}

int scale = this->sceneHeight / maxY < this->sceneWidth / maxX ?

this->sceneHeight / maxY - 1 : this->sceneWidth / maxX - 1;

for (int i = 1; i < vl.length(); i++) {

// TODO[#2]: fix memory leak

QGraphicsTextItem\* item = new QGraphicsTextItem();

item->setPlainText(QString::number(i));

item->setX(vl.at(i).first \* scale);

item->setY(vl.at(i).second \* scale);

qDebug() << "i " << i << " first " << vl.at(i).first << " second "

<< vl.at(i).second << " first scale " << vl.at(i).first \* scale

<< " second scale " << vl.at(i).second \* scale;

this->scene->addItem(item);

}

EdgeList el = graph->*provideEdges*();

for (int i = 0; i < el.length(); i++) {

// TODO[#2]: fix memory leak

QGraphicsLineItem\* item = new QGraphicsLineItem(

vl.at(el.at(i).first).first \* scale,

vl.at(el.at(i).first).second \* scale,

vl.at(el.at(i).second).first \* scale,

vl.at(el.at(i).second).second \* scale

);

this->scene->addItem(item);

}

// ::std::shared\_ptr<QGraphicsTextItem> item =

// ::std::make\_shared<QGraphicsTextItem>();

}

void MainWindow::**removeEdge**(int fv, int sv) {

try {

this->graph->*removeEdge*(fv, sv);

} catch (CanNotRemoveEdgeException& e) {

qDebug() << "catch" << e.*what*();

this->informationLabel->setText(QString(e.*what*()));

return;

} catch (VertexNotFoundException& e) {

qDebug() << "catch" << e.*what*();

this->informationLabel->setText(QString(e.*what*()));

return;

}

this->scene->clear();

this->printGraph();

}

#include <QDebug>

#include "MyTextEdit.h"

MyTextEdit::**MyTextEdit**()

{}

void MyTextEdit::***keyPressEvent***(QKeyEvent \*event) {

if (event->key() == Qt::Key\_Return)

{

QString s = this->text();

QStringList sl = s.split(" ");

if (sl.length() > 2) {

qDebug() << "THIS IS NOT PAIR of vertex";

this->clear();

return;

}

bool ok;

int fv = sl[0].toInt(&ok);

if (!ok) {

qDebug() << sl[0] << " NAN";

this->clear();

return;

}

int sv = sl[1].toInt(&ok);

if (!ok) {

qDebug() << sl[1] << " NAN";

this->clear();

return;

}

qDebug() << fv << sv;

this->clear();

emit removeEdge(fv, sv);

}

else

{

QLineEdit::keyPressEvent(event);

}

}

#include <QDebug>

#include "GraphOnAdjacencyMatrixBuilder.h"

#include "GraphOnAdjacencyMatrix.h"

GraphOnAdjacencyMatrixBuilder::**GraphOnAdjacencyMatrixBuilder**() {

this->vertexList = new VertexList();

// vertexList[0] is zero, because vertex count from 1.

vertexList->append(QPair<int, int>(0, 0));

this->edgeList = new EdgeList();

}

GraphOnAdjacencyMatrixBuilder::~**GraphOnAdjacencyMatrixBuilder**() {

delete this->vertexList;

delete this->edgeList;

}

IGraphBuilder\* GraphOnAdjacencyMatrixBuilder::***addVertex***(

int coordinateX,

int coordinateY

) {

this->vertexList->append(QPair<int, int>(coordinateX, coordinateY));

return this;

}

IGraphBuilder\* GraphOnAdjacencyMatrixBuilder::***addEdge***(

int firstVertex,

int secondVertex

) throw (***VertexNotFoundException***) {

if (firstVertex >= this->vertexList->length()) {

throw VertexNotFoundException(firstVertex);

}

if (secondVertex>= this->vertexList->length()) {

throw VertexNotFoundException(secondVertex);

}

this->edgeList->append(QPair<int, int>(firstVertex, secondVertex));

return this;

}

::std::shared\_ptr<IGraphComplete> GraphOnAdjacencyMatrixBuilder::***build***() throw (

***CanNotBuildAssociatedGraph***, ***VertexNotFoundException***

// TODO[#1]: check is graph associated and hrow exception.

) {

// short\*\* adjacencyMatrix;

::std::vector<::std::vector<short>> adjacencyMatrix(this->vertexList->length());

// qDebug() << "this->vertexList->length()" << this->vertexList->length();

// try {

// adjacencyMatrix = new short\*[this->vertexList->length()];

// } catch (::std::bad\_alloc e) {

// qDebug() << e.what();

// } catch (::std::exception e) {

// qDebug() << e.what();

// } catch (...) {

// qDebug() << "e.what()";

// }

// qDebug() << "ALLOC without e";

for (int rowNumber = 1;

rowNumber < this->vertexList->length();

rowNumber++

) {

// adjacencyMatrix[rowNumber] = new short[this->vertexList->length()];

adjacencyMatrix[rowNumber] = ::std::vector<short>(this->vertexList->length());

for (int columnNumber = 1;

columnNumber < this->vertexList->length();

columnNumber++

) {

adjacencyMatrix[rowNumber][columnNumber] = 0;

}

}

for (int index = 0; index < this->edgeList->length(); index++) {

int firstVertex = this->edgeList->at(index).first;

int secondVertex = this->edgeList->at(index).second;

adjacencyMatrix[firstVertex][secondVertex] = 1;

adjacencyMatrix[secondVertex][firstVertex] = 1;

}

qDebug() << "print adjacencyMatrix";

for (int i = 1; i < this->vertexList->length(); i++) {

for (int j = 1; j < this->vertexList->length(); j++) {

qDebug() << i << " " << j << " " << adjacencyMatrix[i][j];

}

}

// return nullptr;

return ::std::make\_shared<GraphOnAdjacencyMatrix>(

VertexList(\*this->vertexList),

*adjacencyMatrix*

);

}

#include <QDebug>

#include "GraphOnAdjacencyMatrix.h"

GraphOnAdjacencyMatrix::**GraphOnAdjacencyMatrix**(

VertexList vertexList,

::std::vector<::std::vector<short>> edgeMatrix

) {

this->vertexList = vertexList;

this->edgeMatrix = edgeMatrix;

}

GraphOnAdjacencyMatrix::~**GraphOnAdjacencyMatrix**() {}

EdgeList GraphOnAdjacencyMatrix::***provideEdges***() {

EdgeList edgeList;

for (int rowNumber = 1;

rowNumber < this->vertexList.length();

rowNumber++

) {

for (int columnNumber = 0 + rowNumber;

columnNumber < this->vertexList.length();

columnNumber++

) {

if (1 == this->edgeMatrix[rowNumber][columnNumber]) {

edgeList.append(QPair<int, int>(rowNumber, columnNumber));

}

}

}

return edgeList;

}

VertexList GraphOnAdjacencyMatrix::***provideVertex***() {

qDebug() << "provideVertex";

return this->vertexList;

}

void GraphOnAdjacencyMatrix::***removeEdge***(

int firstVertex,

int secondVertex

) throw (***CanNotRemoveEdgeException***, ***VertexNotFoundException***) {

if (firstVertex >= this->vertexList.length()) {

throw VertexNotFoundException(firstVertex);

}

if (secondVertex>= this->vertexList.length()) {

throw VertexNotFoundException(secondVertex);

}

::std::vector<::std::vector<short>> copyOfTheEdgeMatrix =

this->createCopyOfEdgeMatrix();

copyOfTheEdgeMatrix[firstVertex][secondVertex] = 0;

copyOfTheEdgeMatrix[secondVertex][firstVertex] = 0;

if (!graphIsFullConnected(copyOfTheEdgeMatrix)) {

throw CanNotRemoveEdgeException(firstVertex, secondVertex);

}

edgeMatrix[firstVertex][secondVertex] = 0;

edgeMatrix[secondVertex][firstVertex] = 0;

for (int rowNumber = 0;

rowNumber < this->vertexList.length();

rowNumber++

) {

// delete[] copyOfTheEdgeMatrix[rowNumber];

}

// delete[] copyOfTheEdgeMatrix;

}

::std::vector<::std::vector<short>> GraphOnAdjacencyMatrix::**createCopyOfEdgeMatrix**() {

::std::vector<::std::vector<short>> newEdgeMatrix(this->vertexList.length());

for (int rowNumber = 1;

rowNumber < this->vertexList.length();

rowNumber++

) {

newEdgeMatrix[rowNumber] = ::std::vector<short>(this->vertexList.length());

for (int columnNumber = 1;

columnNumber < this->vertexList.length();

columnNumber++

) {

newEdgeMatrix[rowNumber][columnNumber] =

this->edgeMatrix[rowNumber][columnNumber];

}

}

return newEdgeMatrix;

}

// TODO[#1]: use in builder.

bool GraphOnAdjacencyMatrix::**graphIsFullConnected**(::std::vector<::std::vector<short>> edgeMatrix) {

short\* vertexState = new short[this->vertexList.length()];

for (int vertexNumber = 1;

vertexNumber < this->vertexList.length();

vertexNumber++) {

/// 1 - not checked, 2 - need check aka start of edge, 3 - checked

vertexState[vertexNumber] = 1;

}

vertexState[1] = 2;

/// start run

int vertexNumberForCheck;

bool allVertexWasChecked = false;

do {

allVertexWasChecked = true;

for (int vertexNumber = 1;

vertexNumber < this->vertexList.length();

vertexNumber++

) {

if (vertexState[vertexNumber] == 2) {

vertexState[vertexNumber] = 3;

vertexNumberForCheck = vertexNumber;

break;

}

}

for (int endOfEdge = 1;

endOfEdge < this->vertexList.length();

endOfEdge++

) {

if (1 == edgeMatrix[vertexNumberForCheck][endOfEdge]

&& 1 == vertexState[endOfEdge]

) {

vertexState[endOfEdge] = 2;

}

}

for (int vertexNumber = 1;

vertexNumber < this->vertexList.length();

vertexNumber++

) {

if (vertexState[vertexNumber] == 2) {

allVertexWasChecked = false;

}

}

} while (allVertexWasChecked == false);

/// run by vertices for check

bool isFullConnected = true;

for (int vertexNumber = 1;

vertexNumber < this->vertexList.length();

vertexNumber++

) {

if (vertexState[vertexNumber] == 1) {

isFullConnected = false;

}

}

delete[] vertexState;

return isFullConnected;

}

#include <QDebug>

#include "TextFileParser.h"

#include "graph\_interfaces/VertexNotFoundException.h"

TextFileParser::**TextFileParser**(

IGraphBuilder\* iGraphBuilder,

const char\* filePath

) {

this->iGraphBuilder = iGraphBuilder;

this->storage = new QFile(QString(filePath));

}

TextFileParser::**TextFileParser**(

IGraphBuilder\* iGraphBuilder,

::std::string filePath

) {

this->iGraphBuilder = iGraphBuilder;

this->storage = new QFile(QString(filePath.c\_str()));

}

TextFileParser::**TextFileParser**(

IGraphBuilder\* iGraphBuilder,

QString filePath

) {

this->iGraphBuilder = iGraphBuilder;

this->storage = new QFile(filePath);

}

TextFileParser::~**TextFileParser**() {

if (nullptr != this->iGraphBuilder) {

delete this->iGraphBuilder;

this->iGraphBuilder = nullptr;

}

if (nullptr != this->storage) {

delete this->storage;

this->storage = nullptr;

}

}

::std::shared\_ptr<IGraphComplete> TextFileParser::***provideGraph***() throw (

***CanNotParceGraphException***

) {

if (nullptr == this->parsedGraph.get()) {

try {

this->parse();

} catch (CanNotParceGraphException e) {

throw std::move(*e*);

}

}

return parsedGraph;

}

void TextFileParser::***saveGraph***() {

qDebug() << "void TextFileParser::saveGraph() not implemented";

}

void TextFileParser::**parse**() throw(**CanNotParceGraphException**) {

if (!storage->*open*(QIODevice::ReadOnly | QIODevice::Text)) {

qDebug() << "Can not open file withgraph for read.";

throw CanNotParceGraphException(

"Can not open file withgraph for read."

);

}

QTextStream textStream(storage);

bool ok;

/// Parsing vertices.

int vertexCounter = textStream.readLine().toInt(&ok);

if (!ok) {

qDebug() << "Can not read count of vertex.";

throw CanNotParceGraphException(

"Can not read count of vertex."

);

}

for (int vertexNumber = 0; vertexNumber < vertexCounter; vertexNumber++) {

QStringList vertexCoordinatesAsStrings =

textStream.readLine().split(" ");

if (2 != vertexCoordinatesAsStrings.size()) {

::std::ostringstream stringStream;

stringStream << "On vertex #" << vertexNumber + 1

<< " founded " << vertexCoordinatesAsStrings.size()

<< " elements instead 2.";

qDebug() << stringStream.str().c\_str();

throw CanNotParceGraphException(stringStream.str());

}

int coordinateX = vertexCoordinatesAsStrings[0].toInt(&ok);

if (!ok) {

::std::ostringstream stringStream;

stringStream << "On vertex #" << vertexNumber + 1

<< " x-coordinate can not be parse to int.";

qDebug() << stringStream.str().c\_str();

throw CanNotParceGraphException(stringStream.str());

}

int coordinateY = vertexCoordinatesAsStrings[1].toInt(&ok);

if (!ok) {

::std::ostringstream stringStream;

stringStream << "On vertex #" << vertexNumber + 1

<< " y-coordinate can not be parse to int.";

qDebug() << stringStream.str().c\_str();

throw CanNotParceGraphException(stringStream.str());

}

iGraphBuilder->*addVertex*(coordinateX, coordinateY);

}

/// Parsing edges.

int edgeCounter = textStream.readLine().toInt(&ok);

if (!ok) {

qDebug() << "Can not read count of edge.";

throw CanNotParceGraphException(

"Can not read count of edge."

);

}

for (int edgeNumber = 0; edgeNumber < edgeCounter; edgeNumber++) {

QStringList pairOfVerticesAsStrings =

textStream.readLine().split(" ");

if (2 != pairOfVerticesAsStrings.size()) {

::std::ostringstream stringStream;

stringStream << "On edge #" << edgeNumber + 1

<< " founded " << pairOfVerticesAsStrings.size()

<< " elenments instead 2.";

qDebug() << stringStream.str().c\_str();

throw CanNotParceGraphException(stringStream.str());

}

int firstVertex = pairOfVerticesAsStrings[0].toInt(&ok);

if (!ok) {

::std::ostringstream stringStream;

stringStream << "On edge #" << edgeNumber + 1

<< " firstVertex can not be parse to int.";

qDebug() << stringStream.str().c\_str();

throw CanNotParceGraphException(stringStream.str());

}

if (firstVertex > vertexCounter) {

::std::ostringstream stringStream;

stringStream << "On edge #" << edgeNumber + 1

<< " firstVertex is " << firstVertex

<< " but total vertices is " << vertexCounter;

qDebug() << stringStream.str().c\_str();

throw CanNotParceGraphException(stringStream.str());

}

int secondVertex = pairOfVerticesAsStrings[1].toInt(&ok);

if (!ok) {

::std::ostringstream stringStream;

stringStream << "On edge #" << edgeNumber + 1

<< " secondVertex can not be parse to int.";

qDebug() << stringStream.str().c\_str();

throw CanNotParceGraphException(stringStream.str());

}

if (secondVertex > vertexCounter) {

::std::ostringstream stringStream;

stringStream << "On edge #" << edgeNumber + 1

<< " secondVertex is " << firstVertex

<< " but total vertices is " << vertexCounter;

qDebug() << stringStream.str().c\_str();

throw CanNotParceGraphException(stringStream.str());

}

try {

iGraphBuilder->*addEdge*(firstVertex, secondVertex);

} catch (VertexNotFoundException exception) {

throw CanNotParceGraphException(exception.*what*());

}

}

this->parsedGraph = this->iGraphBuilder->*build*();

}

#ifndef MYTEXTEDIT\_H

#define MYTEXTEDIT\_H

#include <QTextEdit>

#include <QLineEdit>

#include <QKeyEvent>

class **MyTextEdit**: public QLineEdit

{

Q\_OBJECT

public:

**MyTextEdit**();

signals:

void **removeEdge**(int, int);

// QWidget interface

protected:

void ***keyPressEvent***(QKeyEvent \*event) override;

};

#endif // MYTEXTEDIT\_H

//class MyTextEdit

//{

// Q\_OBJECT

//public:

// MyTextEdit(QWidget\* parrent = nullptr) {

// }

// void keyPressEvent(QKeyEvent \*event)

// {

// if (event->key() == Qt::Key\_Return)

// {

// qDebug() << this->toPlainText();

// }

// else

// {

// QTextEdit::keyPressEvent(event);

// }

// }

//signals:

// void printed(int, int);

//};

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QGraphicsScene>

#include <QLabel>

#include <QWidget>

#include "graph\_interfaces/IGraphComplete.h"

class **MainWindow** : public QWidget

{

Q\_OBJECT

public:

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

~***MainWindow***();

void **printGraph**();

::std::shared\_ptr<IGraphComplete> graph;

QGraphicsScene\* scene;

QLabel\* informationLabel;

int sceneHeight = 480;

int sceneWidth = 640;

public slots:

void **removeEdge**(int, int);

};

#endif // MAINWINDOW\_H

#ifndef VERTEXNOTFOUNDEXCEPTION\_H

#define VERTEXNOTFOUNDEXCEPTION\_H

#include <exception>

#include <sstream>

class **VertexNotFoundException**: public ::std::exception {

private:

::std::string whatS;

public:

**VertexNotFoundException**(

int vertexNumber

) {

::std::ostringstream stringStream;

stringStream << "Vertex #" << vertexNumber << " not found.";

whatS = stringStream.str();

}

const char\* ***what***() const throw() {

return whatS.c\_str();

}

};

#endif // VERTEXNOTFOUNDEXCEPTION\_H

#ifndef IGRAPHVERTICESPROVIDER\_H

#define IGRAPHVERTICESPROVIDER\_H

#include <QList>

#include <QPair>

typedef QList<QPair<int, int>> VertexList;

class **IGraphVerticesProvider** {

public:

virtual VertexList ***provideVertex***() = 0;

};

#endif // IGRAPHVERTICESPROVIDER\_H

#ifndef IGRAPHVERTEXASSOCIATEDADDER\_H

#define IGRAPHVERTEXASSOCIATEDADDER\_H

#include <QPoint>

#include "graph\_interfaces/VertexNotFoundException.h"

class **IGraphVertexAssociatedAdder** {

public:

virtual void ***addVertex***(

QPoint const& coordinates,

int associatedVertex

) throw(***VertexNotFoundException***) = 0;

virtual void ***addVertex***(

int coordinateX,

int coordinateY,

int associatedVertex

) throw(***VertexNotFoundException***) = 0;

};

#endif // IGRAPHVERTEXASSOCIATEDADDER\_H

#ifndef IGRAPHPROVIDER\_H

#define IGRAPHPROVIDER\_H

#include "graph\_interfaces/IGraphEdgesProvider.h"

#include "graph\_interfaces/IGraphVerticesProvider.h"

class **IGraphProvider**:

public IGraphVerticesProvider,

public IGraphEdgesProvider {};

#endif // IGRAPHPROVIDER\_H

#ifndef IGRAPHPARSER\_H

#define IGRAPHPARSER\_H

#include <exception>

#include <memory>

#include <sstream>

#include "graph\_interfaces/IGraphComplete.h"

class **CanNotParceGraphException**: public ::std::exception {

private:

::std::string whatS;

public:

**CanNotParceGraphException**(::std::string reason) {

::std::ostringstream stringStream;

stringStream << "Can not parse graph because " << reason << ".";

whatS = stringStream.str();

}

const char\* ***what***() const throw() {

return whatS.c\_str();

};

};

class **IGraphParser** {

public:

virtual ::std::shared\_ptr<IGraphComplete> ***provideGraph***() throw (

***CanNotParceGraphException***

)= 0;

virtual void ***saveGraph***() = 0;

};

#endif // IGRAPHPARSER\_H

#ifndef IGRAPHEDGESPROVIDER\_H

#define IGRAPHEDGESPROVIDER\_H

#include <QList>

#include <QPair>

typedef QList<QPair<int, int>> EdgeList;

class **IGraphEdgesProvider** {

public:

virtual EdgeList ***provideEdges***() = 0;

};

#endif // IGRAPHEDGESPROVIDER\_H

#ifndef IGRAPHEDGEREMOVER\_H

#define IGRAPHEDGEREMOVER\_H

#include <exception>

#include <sstream>

#include "graph\_interfaces/VertexNotFoundException.h"

class **CanNotRemoveEdgeException**: public ::std::exception {

private:

::std::string whatS;

public:

**CanNotRemoveEdgeException**(

int firstVertex,

int secondVertex

) {

::std::ostringstream stringStream;

stringStream << "It is not possible to remove the edge between vertices"

<< "#" << firstVertex << " and #" << secondVertex

<< "because this will make the graph incoherent.";

whatS = stringStream.str();

}

const char\* ***what***() const throw() {

return whatS.c\_str();

}

};

class **IGraphEdgeRemover** {

public:

virtual void ***removeEdge***(

int firstVertex,

int secondVertex

) throw(***CanNotRemoveEdgeException***, ***VertexNotFoundException***) = 0;

};

#endif // IGRAPHEDGEREMOVER\_H

#ifndef IGRAPHCOMPLETE\_H

#define IGRAPHCOMPLETE\_H

#include "graph\_interfaces/IGraphEdgeRemover.h"

#include "graph\_interfaces/IGraphProvider.h"

class **IGraphComplete**: public IGraphEdgeRemover, public IGraphProvider {};

#endif // IGRAPHCOMPLETE\_H

#ifndef IGRAPHBUILDER\_H

#define IGRAPHBUILDER\_H

#include <exception>

#include <memory>

#include "graph\_interfaces/IGraphComplete.h"

#include "graph\_interfaces/VertexNotFoundException.h"

class **CanNotBuildAssociatedGraph**: public ::std::runtime\_error {

public:

const char\* ***what***() const throw() {

return "Can not build graph, because he in not associated.";

}

};

class **IGraphBuilder** {

public:

virtual IGraphBuilder\* ***addVertex***(int coordinateX, int coordinateY) = 0;

virtual IGraphBuilder\* ***addEdge***(

int firstVertex,

int secondVertex

) throw(***VertexNotFoundException***) = 0;

virtual ::std::shared\_ptr<IGraphComplete> ***build***() throw (

***CanNotBuildAssociatedGraph***, ***VertexNotFoundException***

) = 0;

};

#endif // IGRAPHBUILDER\_H

#ifndef GRAPHBUILDERIMPLEMENTATION\_H

#define GRAPHBUILDERIMPLEMENTATION\_H

#include "graph\_interfaces/IGraphBuilder.h"

#include "graph\_interfaces/IGraphEdgesProvider.h"

#include "graph\_interfaces/IGraphVerticesProvider.h"

class **GraphOnAdjacencyMatrixBuilder**: public IGraphBuilder

{

private:

/// vertexList[0] is zero, because vertex count from 1.

VertexList\* vertexList;

EdgeList\* edgeList;

public:

**GraphOnAdjacencyMatrixBuilder**();

~**GraphOnAdjacencyMatrixBuilder**();

// IGraphBuilder interface

public:

IGraphBuilder\* ***addVertex***(int coordinateX, int coordinateY) override;

IGraphBuilder\* ***addEdge***(int firstVertex, int secondVertex) throw(

***VertexNotFoundException***

) override;

::std::shared\_ptr<IGraphComplete> ***build***() throw (

***CanNotBuildAssociatedGraph***, ***VertexNotFoundException***

) override;

};

#endif // GRAPHBUILDERIMPLEMENTATION\_H

#ifndef GRAPHCOMPLETEIMPLEMENTATION\_H

#define GRAPHCOMPLETEIMPLEMENTATION\_H

#include "graph\_interfaces/IGraphComplete.h"

class **GraphOnAdjacencyMatrix**: public IGraphComplete

{

private:

/// vertexList[0] is zero, because vertex count from 1.

VertexList vertexList;

/// edgeMatrix's first line and first column are empty for convenience.

::std::vector<::std::vector<short>> edgeMatrix;

public:

**GraphOnAdjacencyMatrix**(

VertexList vertexList,

::std::vector<::std::vector<short>> edgeMatrix

);

~**GraphOnAdjacencyMatrix**();

private:

::std::vector<::std::vector<short>> **createCopyOfEdgeMatrix**();

bool **graphIsFullConnected**(::std::vector<::std::vector<short>> edgeMatrix);

// IGraphEdgesProvider interface

public:

EdgeList ***provideEdges***() override;

// IGraphVerticesProvider interface

public:

VertexList ***provideVertex***() override;

// IGraphEdgeRemover interface

public:

void ***removeEdge***(int firstVertex, int secondVertex) throw(

***CanNotRemoveEdgeException***, ***VertexNotFoundException***

) override;

};

#endif // GRAPHCOMPLETEIMPLEMENTATION\_H

#ifndef TEXTFILEPARSER\_H

#define TEXTFILEPARSER\_H

#include <QFile>

#include <string>

#include "graph\_interfaces/IGraphBuilder.h"

#include "graph\_interfaces/IGraphParser.h"

class **TextFileParser**: public IGraphParser {

private:

IGraphBuilder\* iGraphBuilder;

QFile\* storage;

::std::shared\_ptr<IGraphComplete> parsedGraph;

public:

**TextFileParser**(IGraphBuilder\* iGraphBuilder, const char\* filePath);

**TextFileParser**(IGraphBuilder\* iGraphBuilder, ::std::string filePath);

**TextFileParser**(IGraphBuilder\* iGraphBuilder, QString filePath);

~**TextFileParser**();

private:

void **parse**() throw(**CanNotParceGraphException**);

// IGraphParser interface

public:

::std::shared\_ptr<IGraphComplete> ***provideGraph***() throw(

***CanNotParceGraphException***

) override;

void ***saveGraph***() override;

};

#endif // TEXTFILEPARSER\_H