Санкт-Петербургский Государственный Электротехнический Университет
«ЛЭТИ» им. В. И. Ульянова (Ленина)

Кафедра информационных систем

Отчет

По практической работе №8

По дисциплине «Объектно-ориентированное программирование»

Студенты группы 2372 _____ Тубшинов В. Т., Алексеев Г.

Преподаватель _____ Егоров С. С.

г. Санкт-Петербург

2023 г.

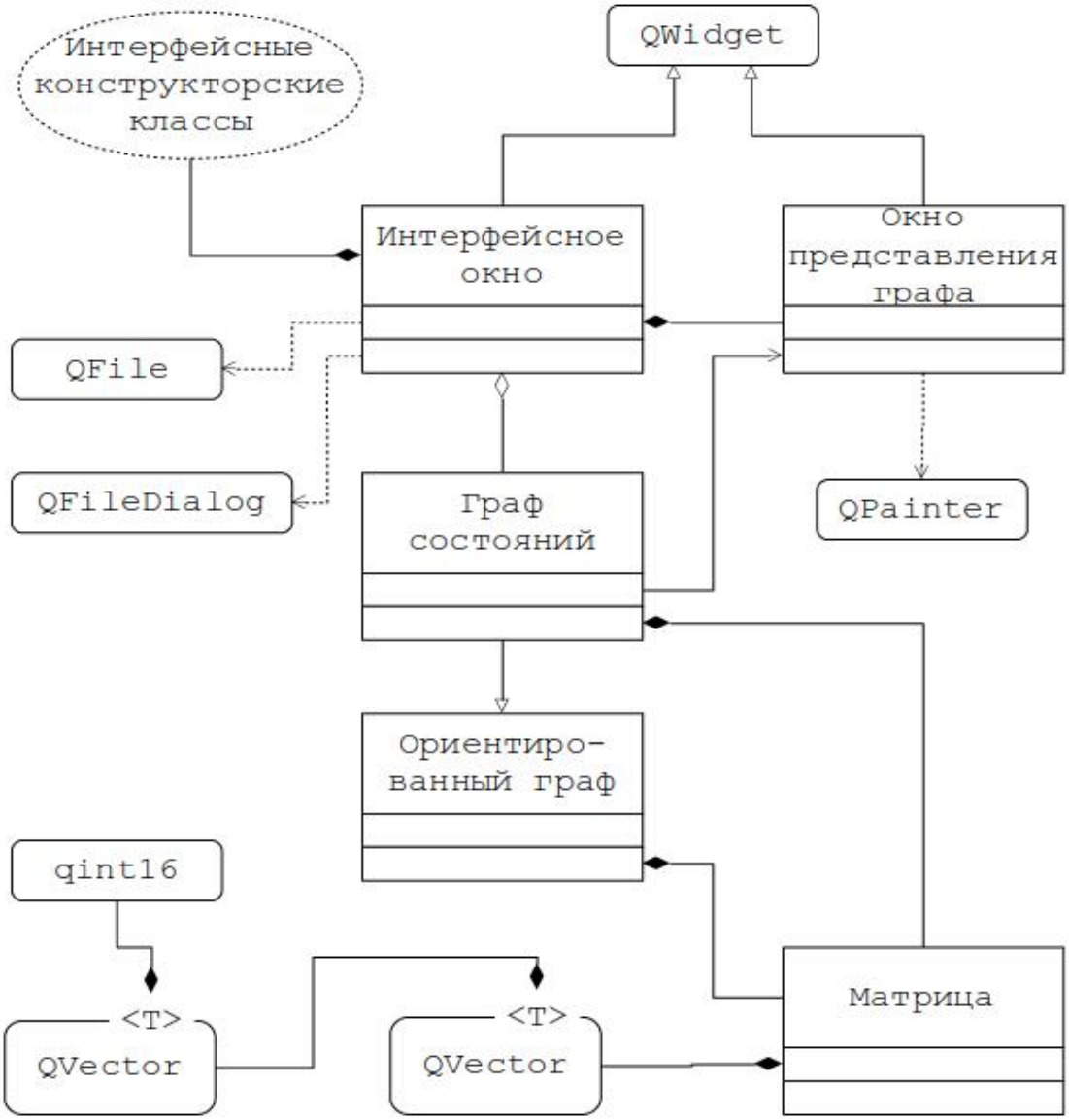


Рис.8. Диаграмма классов работы №8

Разработать GUI приложение, выполняющее функцию визуализации графа состояний.

Граф состояний - это ориентированный граф, одна из вершин которого в каждый момент времени считается активной. Каждой дуге приписано некоторое событие, при возникновении которого происходит смена активной вершины.

Граф состояний описывается матрицей, число строк которой равно числу вершин, а число столбцов - числу событий. Элементом i-

ой строки и j-го столбца является номер строки (т.е. соответствующая ей вершина графа), которая становится активной при возникновении j-го события, если при этом вершина i была активна.

Основной функцией объекта класса "Интерфейсное окно" является выбор файла, который содержит данные о графе состояний. При чтении файла необходимо проверить корректность данных и в случае обнаружения ошибки необходимо сформировать соответствующее сообщение пользователю.

Номер активной вершины также задается в интерфейсе.

При корректности данных создается объект класса "Граф состояний", устанавливаются (если необходимо) связи между новым объектом и существующими, после чего граф отображается в соответствующем окне (объект класса "Окно представления графа").

Активная вершина помечается цветом. При смене значения номера активной вершины должны происходить изменения в отображении.

В интерфейсе должна быть предусмотрена возможность инициирования любого из возможных событий. При их возникновении должен происходить переход в новую активную вершину, согласно графу, смена значения в интерфейсном окне и его перерисовка.

При выборе в интерфейсе другого графа (другого файла) старый должен заменяться на новый, номер активной вершины принимать исходное (корректное) значение и граф перерисовываться.

Реализовать и отладить программу, удовлетворяющую сформулированным требованиям и заявленным целям. Разработать контрольные примеры и оттестировать на них программу. Оформить отчет, сделать выводы по работе.

Спецификации классов:

Класс Graph:

Атрибуты:

private:

vector<Node*> nodes - вектор содержащий вершины графа;

public:

int count - счетчик

Методы:

public:

Graph() - конструктор класса;

vector<Node*> getNodes() const - возвращает ссылку на вершину;

int size() const - возвращает количество вершин;

bool empty() - возвращает 1 - если вершин нет и 0 - если есть;

void addNode(char) - метод добавления вершины;

void addEdge(Node*, Node*) - метод добавления ребра;

string selectFile() - выбор входного файла;

short InputFileValues(string) - метод считывания данных с файла;

Класс Interface:

Атрибуты:

private:

Q_OBJECT

QPushButton *ChangeBtnFile - кнопка для выбора файла;

Sample *sample;

Derivesample* derivesample;

Методы:

public:

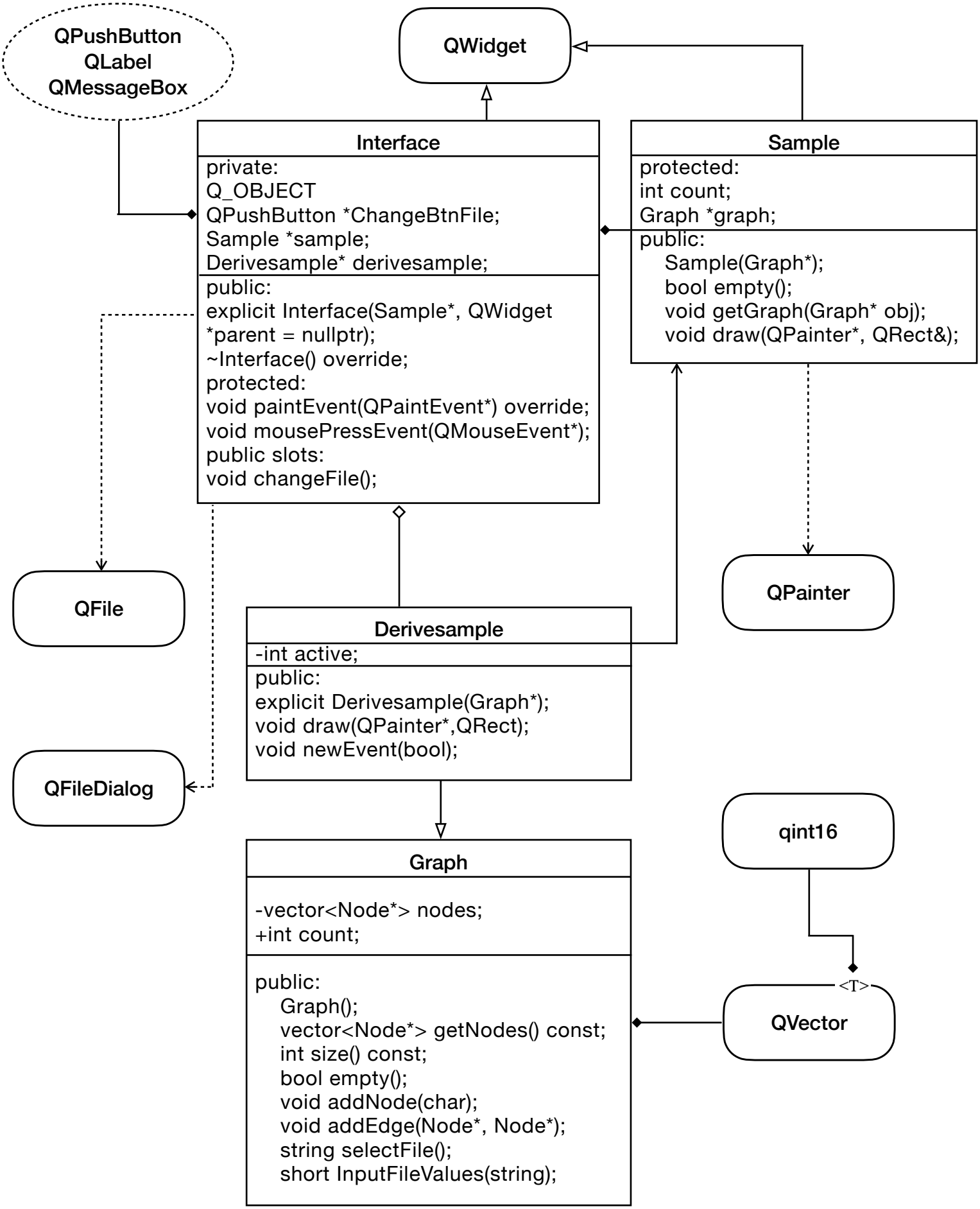
explicit Interface(Sample*, QWidget *parent = nullptr);

~Interface() override - деструктор класса;

protected:
void mousePressEvent(QMouseEvent*);
void paintEvent(QPaintEvent*) override - вывод на экран;
public slots:
void changeFile() - Выбор другого файла;

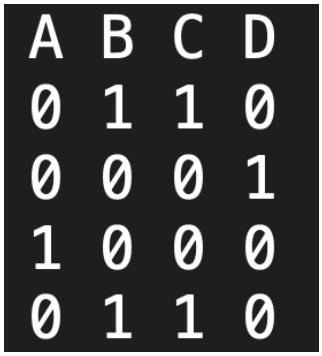
Класс Sample:
Атрибуты:
protected:
int count - счетчик;
Graph *graph - граф;
Методы:
public:
Sample(Graph*) - конструктор класса;
bool empty() - возвращает 1 - если вершин нет и 0 - если есть;
void getGraph(Graph* obj) - функция обращающаяся к атрибуту graph;
void draw(QPainter*, QRect&) - вывод графа на экран;

Класс Derivesample:
Атрибуты:
private:
int active - параметр, описывающий состояние вершины;
Методы:
public:
explicit Derivesample(Graph*) - граф состояний;
void draw(QPainter*,QRect) - вывод на экран;
void newEvent(bool) - изменяет параметр active в зависимости от действий пользователя;


QPushButton
QLabel
QMessageBox
QWidget
Interface
private:
Q_OBJECT
QPushButton *ChangeBtnFile;
Sample *sample;
Derivesample* derivesample;
public:
explicit Interface(Sample*, QWidget *parent = nullptr);
~Interface() override;
protected:
void paintEvent(QPaintEvent*) override;
void mousePressEvent(QMouseEvent*);
public slots:
void changeFile();
Sample
protected:
int count;
Graph *graph;
public:
Sample(Graph*);
bool empty();
void getGraph(Graph* obj);
void draw(QPainter*, QRect&);
QFile
QPainter
Derivesample
-int active;
public:
explicit Derivesample(Graph*);
void draw(QPainter*,QRect);
void newEvent(bool);
QFileDialog
qint16
Graph
-vector<Node*> nodes;
+int count;
public:
Graph();
vector<Node*> getNodes() const;
int size() const;
bool empty();
void addNode(char);
void addEdge(Node*, Node*);
string selectFile();
short InputFileValues(string);
<T>
QVector

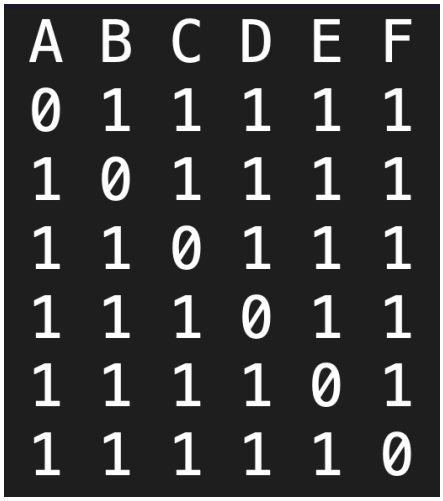
Входные файлы(матрицы смежности):

input1.txt



```
A B C D
0 1 1 0
0 0 0 1
1 0 0 0
0 1 1 0
```

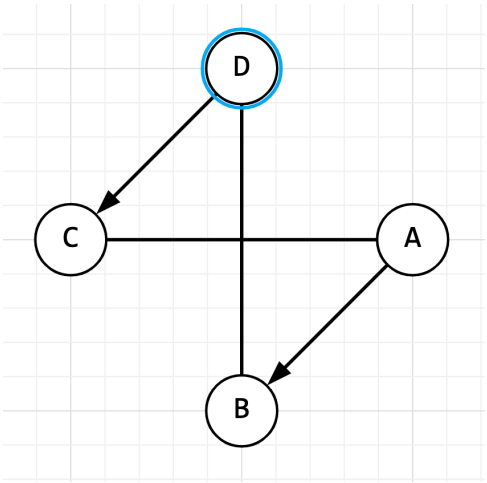
input2.txt

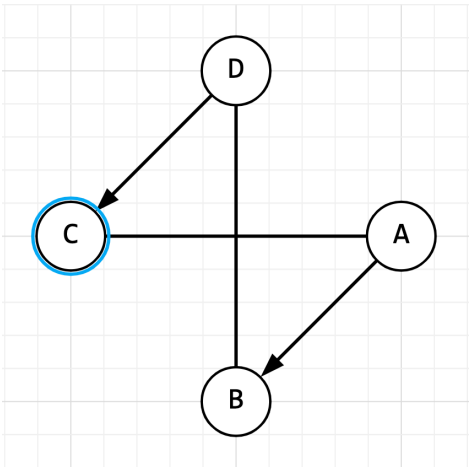


```
A B C D E F
0 1 1 1 1 1
1 0 1 1 1 1
1 1 0 1 1 1
1 1 1 0 1 1
1 1 1 1 0 1
1 1 1 1 1 0
```

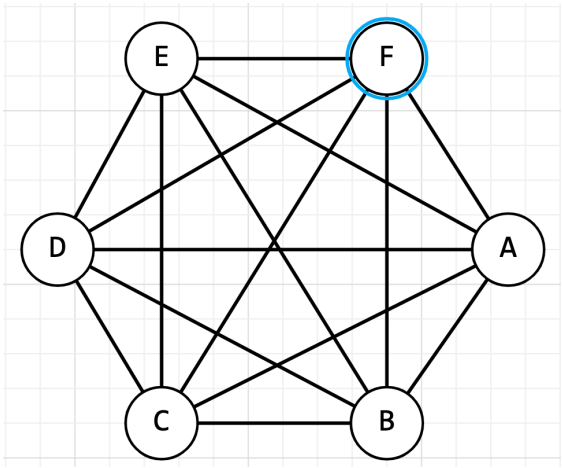
Ожидаемые данные:

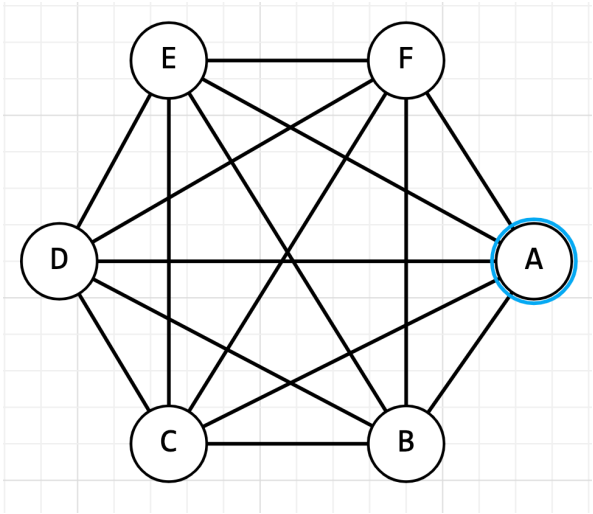
Для входного файла «input1.txt»



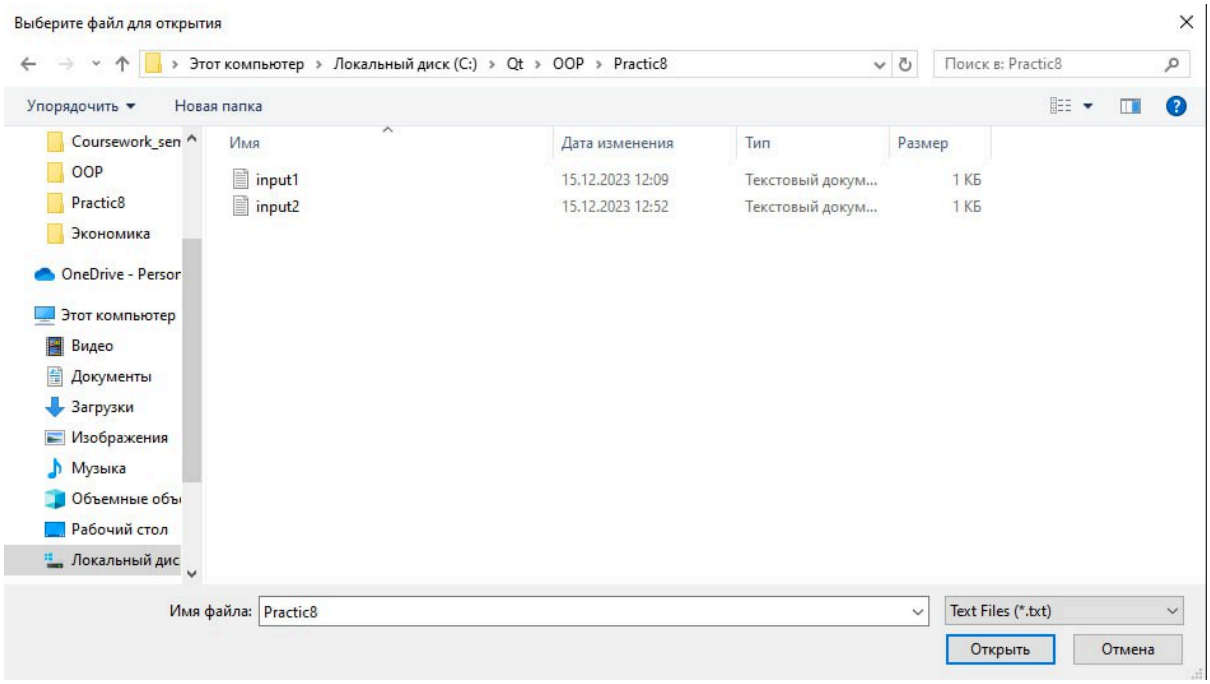


Для входного файла «input2.txt»

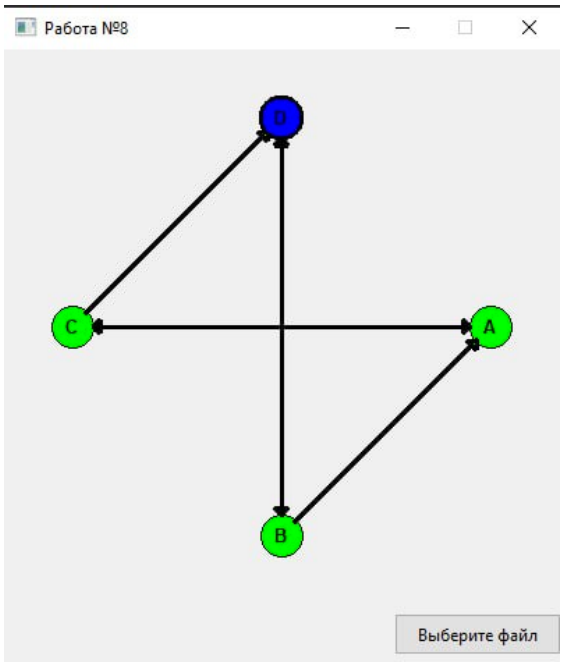


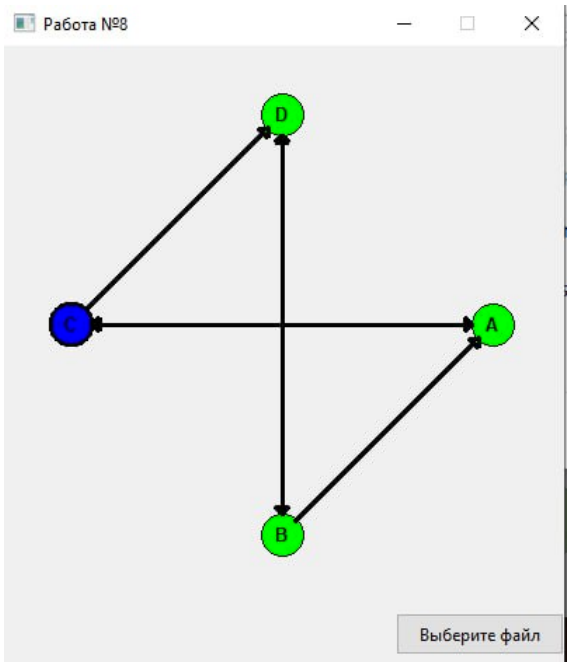


Работа программы:

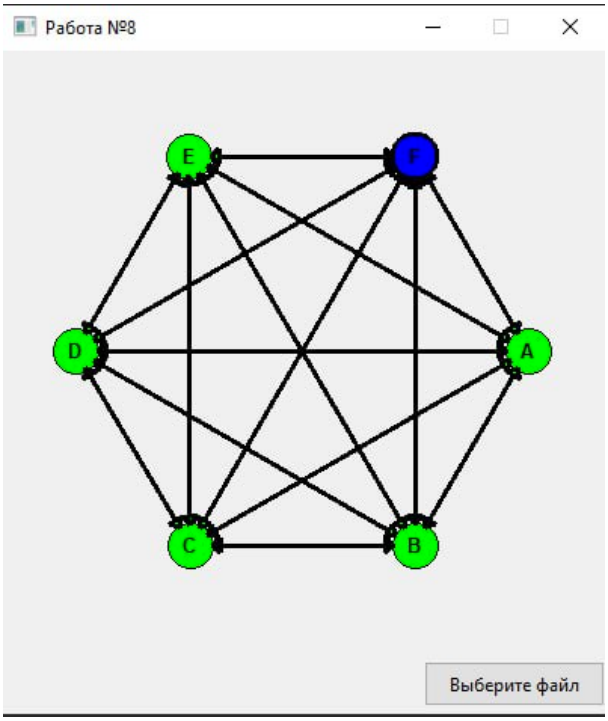


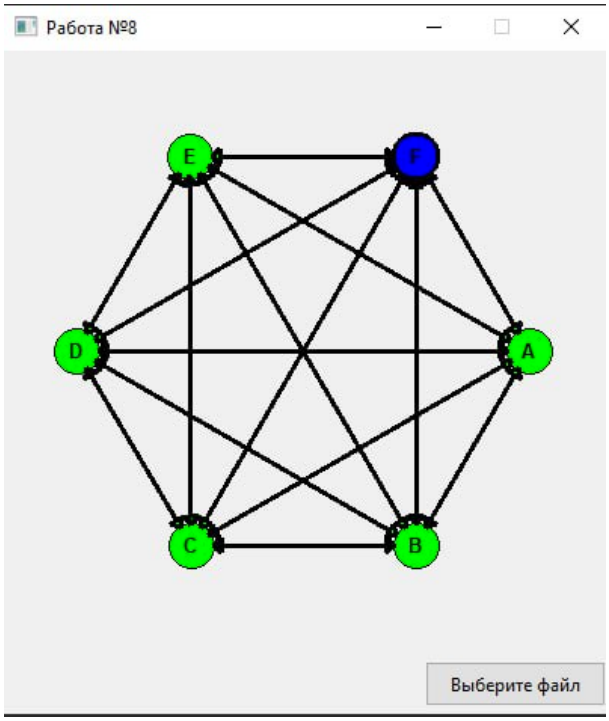
Для входного файла «input1.txt»





Для входного файла «input2.txt»





Вывод: В ходе работы было разработано GUI приложение для визуализации графа состояний. Пользователь выбирает файл с данными о графе, при чтении файла проверяется корректность данных. Если данные корректны, создаётся объект класса "Граф состояний", затем граф передаётся окну представления и визуализируется. Активная вершина помечается цветом. При смене значения номера активной вершины происходят изменения в отображении.