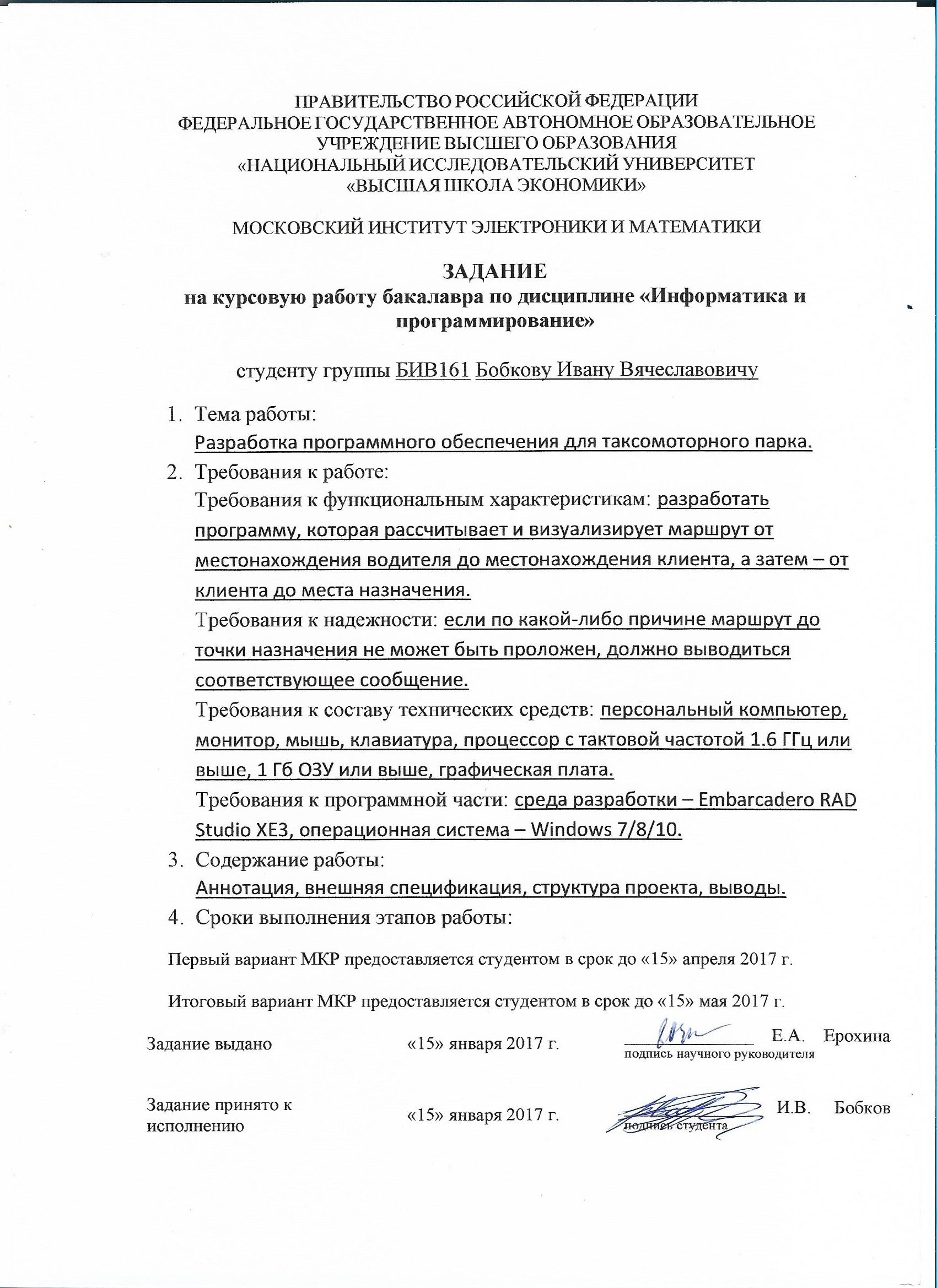
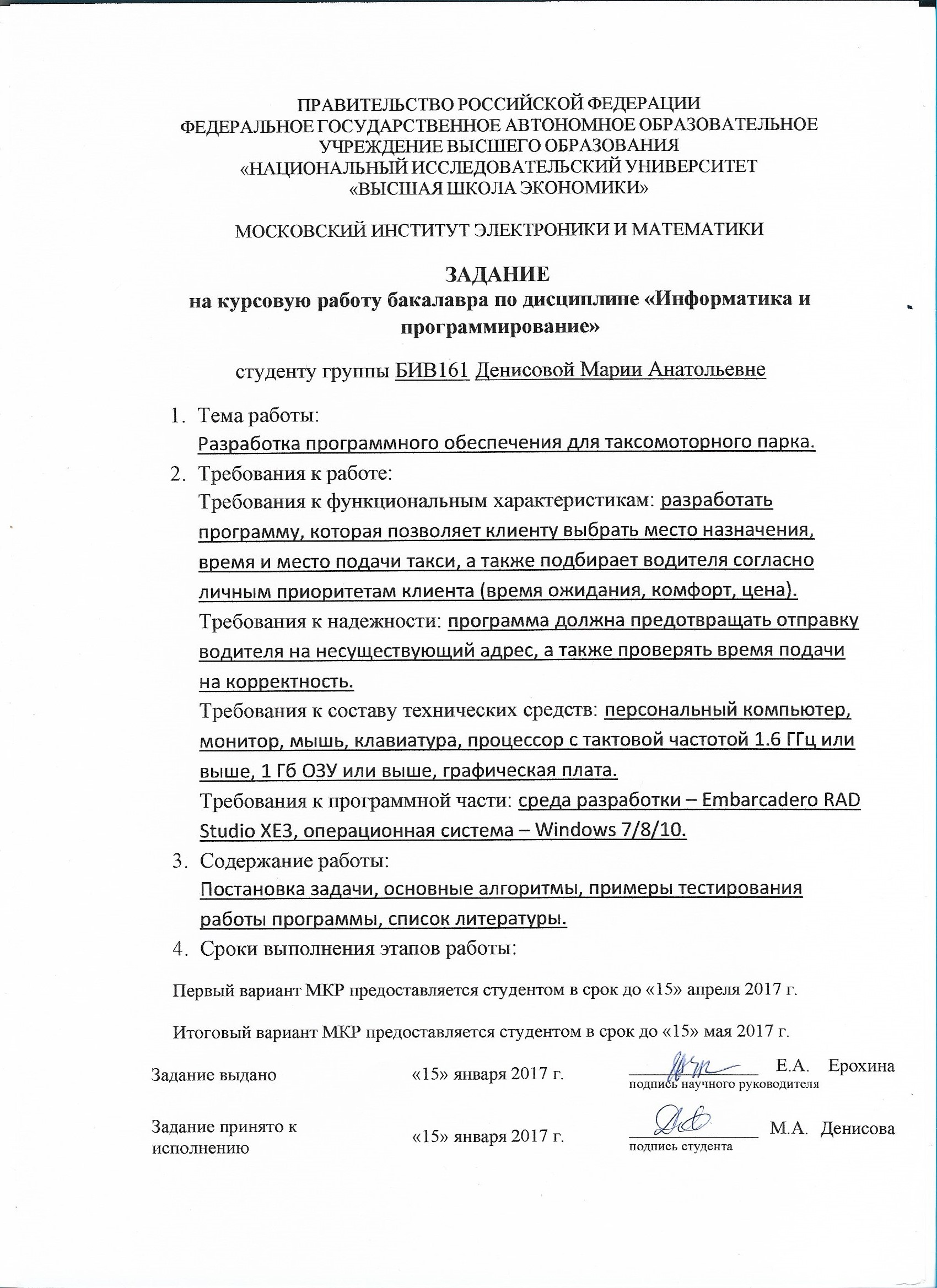
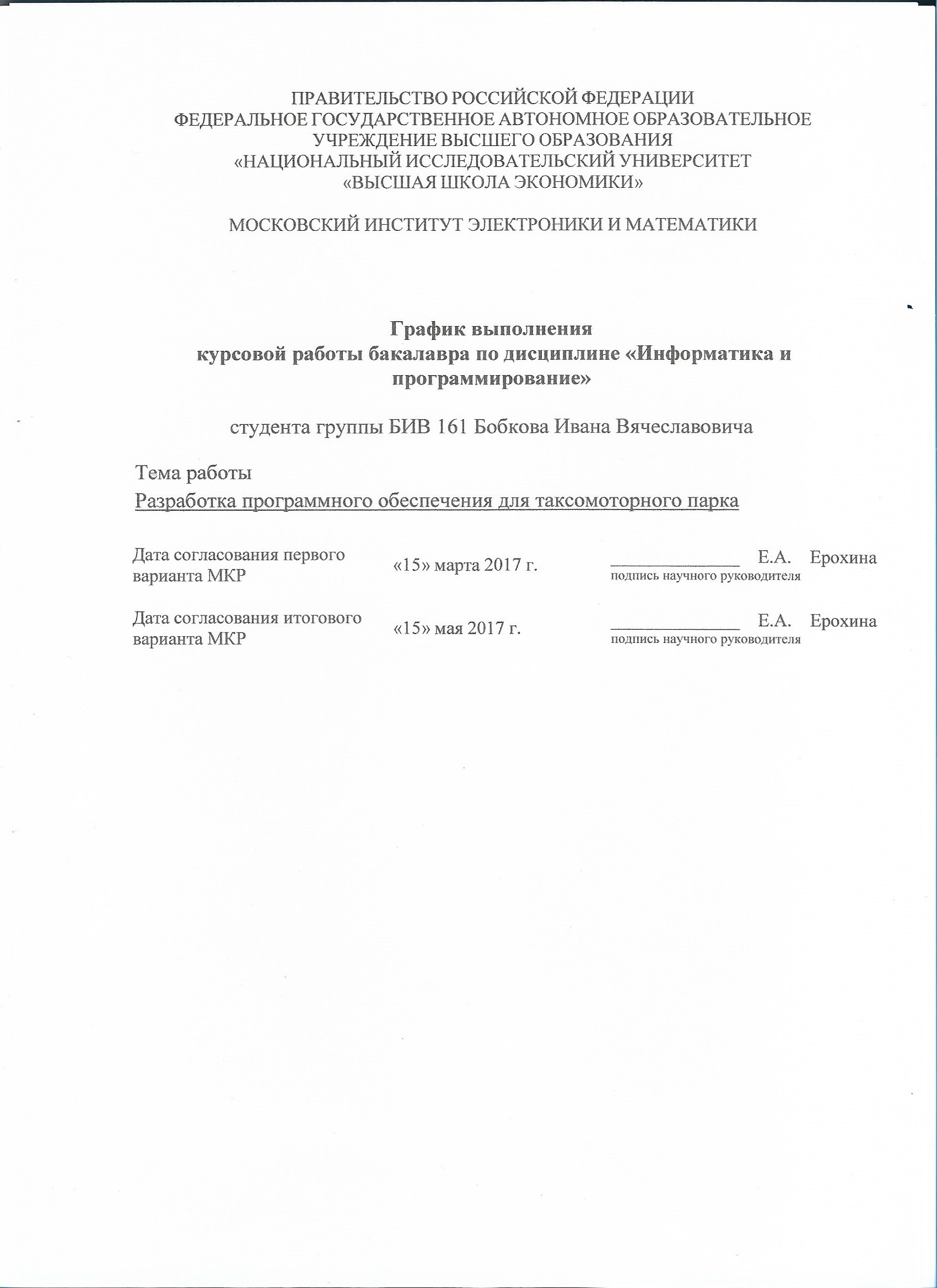
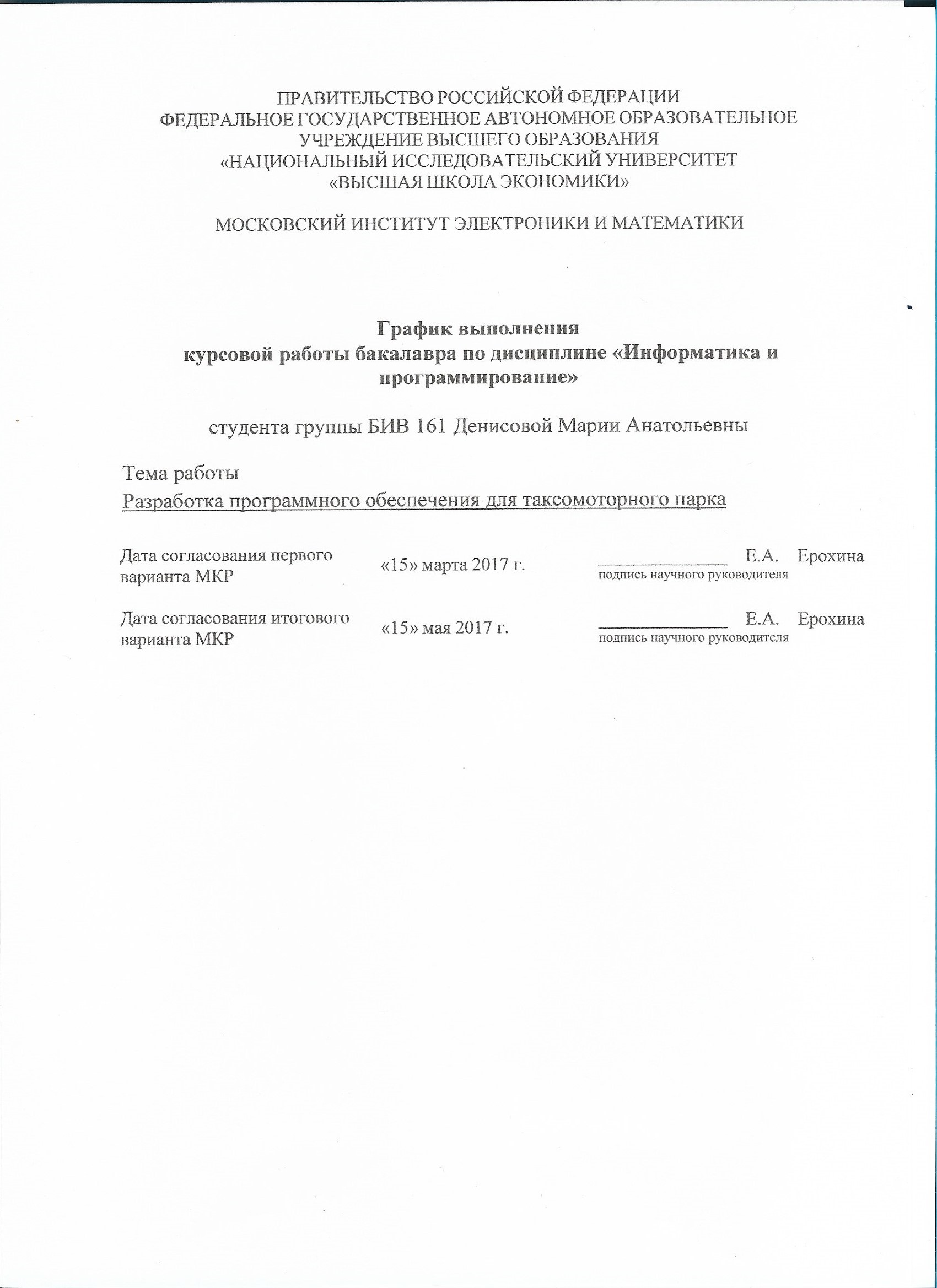
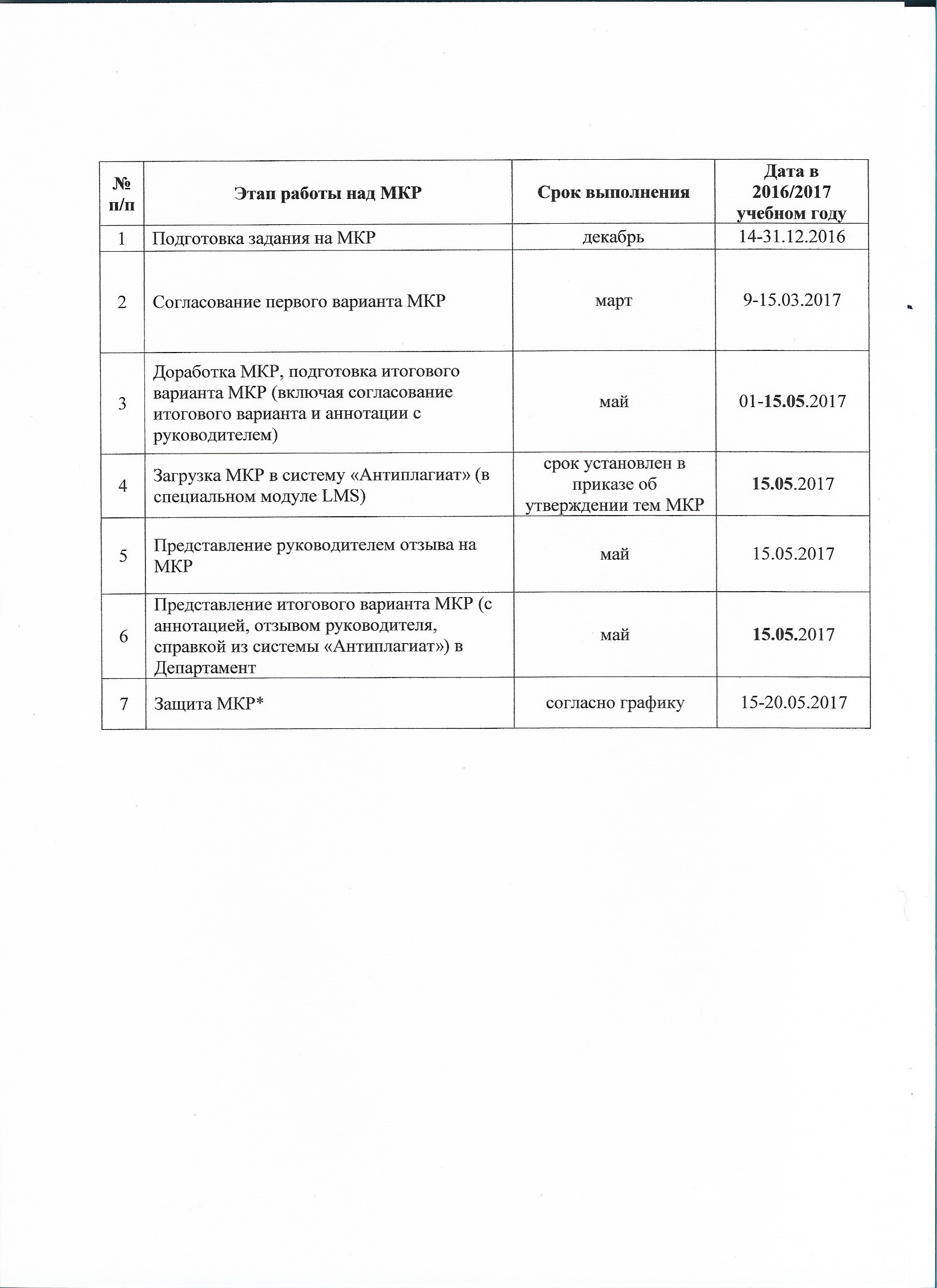
****

****

****

****

****

****

**ОГЛАВЛЕНИЕ**

# **АННОТАЦИЯ**

В данной курсовой работе описаны этапы разработки программного обеспечения для службы такси. ПО написано на языке Object Pascal c использованием СУБД MySQL, где хранятся данные о водителях, автомобилях и заказах, а также карта города. Программа работает в трех режимах: «Водитель», «Диспетчер» и «Пассажир». В режиме «Водитель» программа выводит на экран из базы данных текущий заказ водителя, позволяет ему завершить или отменить заказ, а также построить кратчайший маршрут к точке назначения. В режиме «Пассажир» программа позволяет пользователю заказать такси в соответствии с желаемыми параметрами (адрес, комфорт, количество пассажиров). Режим «Диспетчер» выводит на экран информацию, хранящуюся в базе данных.

За результат работы программы принимаются следующие параметры: рассчитанная стоимость заказа, минимально возможное время в пути до точки назначения, данные подобранного водителя (ФИО, автомобиль, госномер), время ожидания такси, ориентировочное время прибытия, а также визуализированный маршрут, построенный до точки назначения.

В перспективе эту программу могли бы использовать развивающиеся организации, предоставляющие услуги по перевозке пассажиров, которые хотели бы принимать заказы не только по телефону, но и через интернет.

The present course work provides information about taxi service software development. The software consists of an Object Pascal program using MySQL where data (automobiles, drivers, orders) and map are stored. The program works in three modes: “Driver”, “Dispatcher” and “Passenger”. “Driver” mode shows an available order. Here a driver can cancel and complete orders. The program builds and shows the shortest path to a destination point. In “Passenger” mode the program allows to order a cab depending on chosen parameters (address, comfort, number of passengers). “Dispatcher” mode can be used to look through information in the database.

The following parameters should be considered as a result: calculated price, minimal time to a destination point and information about the assigned driver (initials, automobile, registration sign).

This program can be used by developing taxi cab companies which want to improve their service by accepting orders from the Internet.

# **3. УСЛОВИЕ ЗАДАЧИ**

Написать программу, которая подбирает водителя согласно заданным параметрам (адрес, комфорт, количество пассажиров, дополнительные опции), а также строит по точкам кратчайший маршрут от местонахождения водителя до клиента (пользователя) и от клиента до места назначения.

# **4. ПОСТАНОВКА ЗАДАЧИ**

**4.1. Построение кратчайшего маршрута**

Дано:

* driver\_location (id местоположения водителя) – цел
* start (id местоположения клиента) – цел
* finish (id точки назначения) – цел
* edges [1:157] {first\_node, second\_node, distance} – таблица базы данных [строки] {поля таблицы}
* A [1:100][1:100] (матрица смежности – номера столбцов и строк соответствуют номерам вершин, а значения соответствуют расстоянию между двумя вершинами: 0 – вершины совпадают, 9999 – дороги нет) – цел

Результат:

* Route1 [1:k] (маршрут от водителя к пассажиру) – цел
* Route2 [1:m] (маршрут от пассажира к пункту назначения)– цел

При:

* 1 ≤ driver\_location ≤ 100
* 1 ≤ start ≤ 100
* 1 ≤ finish ≤ 100

Связь:

∀x = ,∀y =

A[x][y] = A[y][x] =

Route1[1] = driver\_location

Route1[k] = start

∀t1 = 0 < A [ Route1[t1-1] ] [ Route1[t1] ] < 9999

∄a[1:n].: a[1] = driver\_location,

a[n] = start,

<

Route2[1] = start

Route2[m] = finish

∀t2 = 0 < A [ Route2[t2-1] ] [ Route2[t2] ] < 9999

∄a[1:n].: a[1] = start,

a[n] = finish,

<

**4.2. Подбор водителя**

Дано:

* tripTime(время кратчайшего маршрута от пассажира до точки назначения) – цел
* wide\_trunk (наличие большого багажника) – цел
* baby\_seat (количество детских кресел) – цел
* passengers (количество пассажиров) – цел
* comfort\_rate (уровень комфорта) – цел
* dist [1:25] (дистанция в минутах от пассажира до каждого водителя) – цел
* now (текущее время из операционной системы) – время
* driver\_search [1:25] {id, available, wide\_trunk, baby\_seat, passengers, comfort\_rate} – таблица базы данных [строки] {поля таблицы} (available – доступность водителя)

Результат:

* id (идентификатор водителя) - цел
* time (время ожидания подачи машины в минутах) - цел
* finish\_time (ориентировочное время прибытия) – время

или

* сообщение «Нет подходящего водителя»

При:

* tripTime ∈ ℕ
* 0 ≤ wide\_trunk  ≤ 1
* 0 ≤ baby\_seat ≤ 3
* 1 ≤ passengers ≤ 8
* 1 ≤ comfort\_rate ≤ 5
* dist[1:25] ∈ ℕ

Связь:

∃T: ∀t ∈ T 1 ≤ t ≤ 25,

driver\_search [t] {available} = 1,

driver\_search [t] {wide\_trunk} = wide\_trunk,

driver\_search [t] {baby\_seat} ≥ baby\_seat,

driver\_search [t] {passengers} ≥ passengers,

| driver\_search [t] {comfort\_rate} – comfort\_rate | ≤ 1

∃tmin: tmin ∈ T,

∀n ∈ T n ≠ tmin dist [driver\_search [tmin] {id}] ≥ dist [driver\_search [n] {id}]

id = driver\_search [tmin] {id}

time = dist [driver\_search [tmin] {id}]

finish\_time = now + time + tripTime

# **5. ВНЕШНЯЯ СПЕЦИФИКАЦИЯ**

**5.1. Главное меню**

В главном меню программы (рис. 1) пользователю предлагается выбрать один из трех режимов для дальнейшей работы: «Водитель», «Диспетчер» или «Пассажир». Также в правом верхнем углу окна расположены сервисные кнопки («i», «?», «x»).



рис. 1. Главное меню

Сервисная кнопка « i » (information) выводит на экран окно (рис. 2), содержащее информацию о разработчиках данной программы.

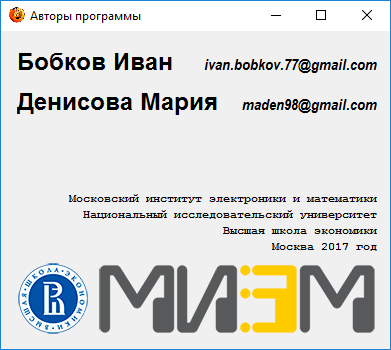


рис. 2. Авторы программы

С помощью сервисной кнопки « ? » (help) пользователь может получить информацию о том, как пользоваться программой. Информация выводится в новом окне (рис. 3). Для того чтобы получить помощь в работе с программой, необходимо нажать в отобразившемся окне оранжевую кнопку с названием желаемого режима, после чего информация о нем отобразится на экране.

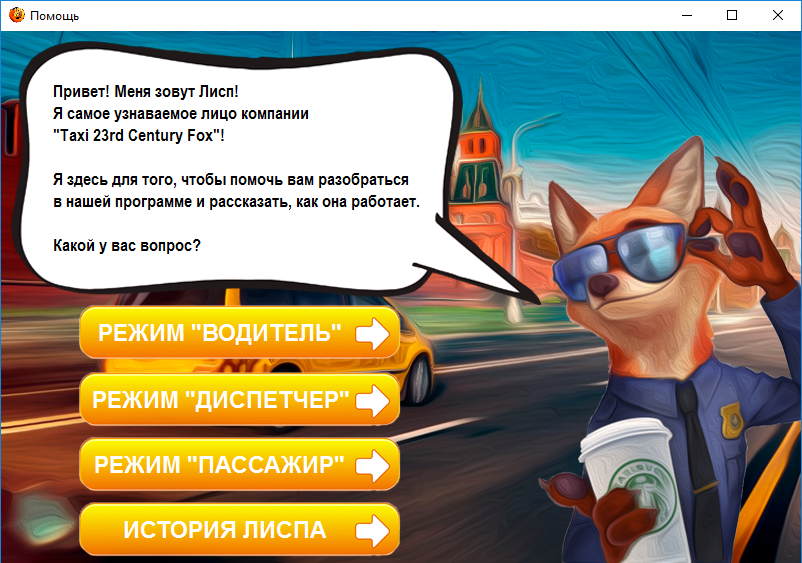


рис. 3. Помощь

С помощью кнопки « x » осуществляется выход из программы.

При нажатии на кнопку «Водитель» появляется окно (рис. 4), где пользователь (водитель) выбирает нужную фамилию из выпадающего списка, а также вводит единый для всех водителей пароль “miem” для дальнейшего доступа к программе.

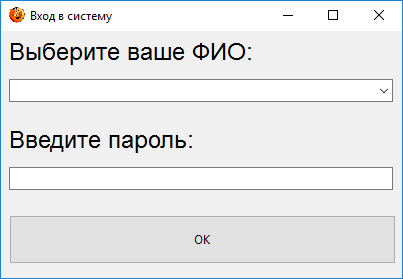
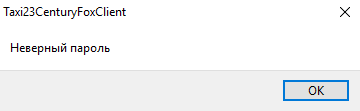


рис. 4. Вход в режим «Водитель»

При вводе неверного пароля программа выведет на экран сообщение (рис. 5), запрещающее дальнейший доступ к режиму.

рис. 5. Сообщение «Неверный пароль»

При нажатии на кнопку «Диспетчер» на экране всплывает окно (рис. 6), запрашивающее единый код для дальнейшего доступа к режиму.

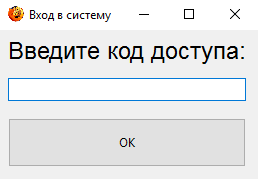


рис. 6. Вход в режим «Диспетчер»

При вводе пользователем (диспетчером) кода “2323” доступ к режиму «Диспетчер» будет разрешен. В случае ввода другой комбинации символов на экране появится сообщение о неправильности введенного кода (рис. 7).

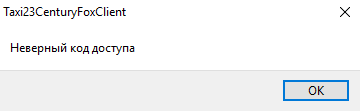
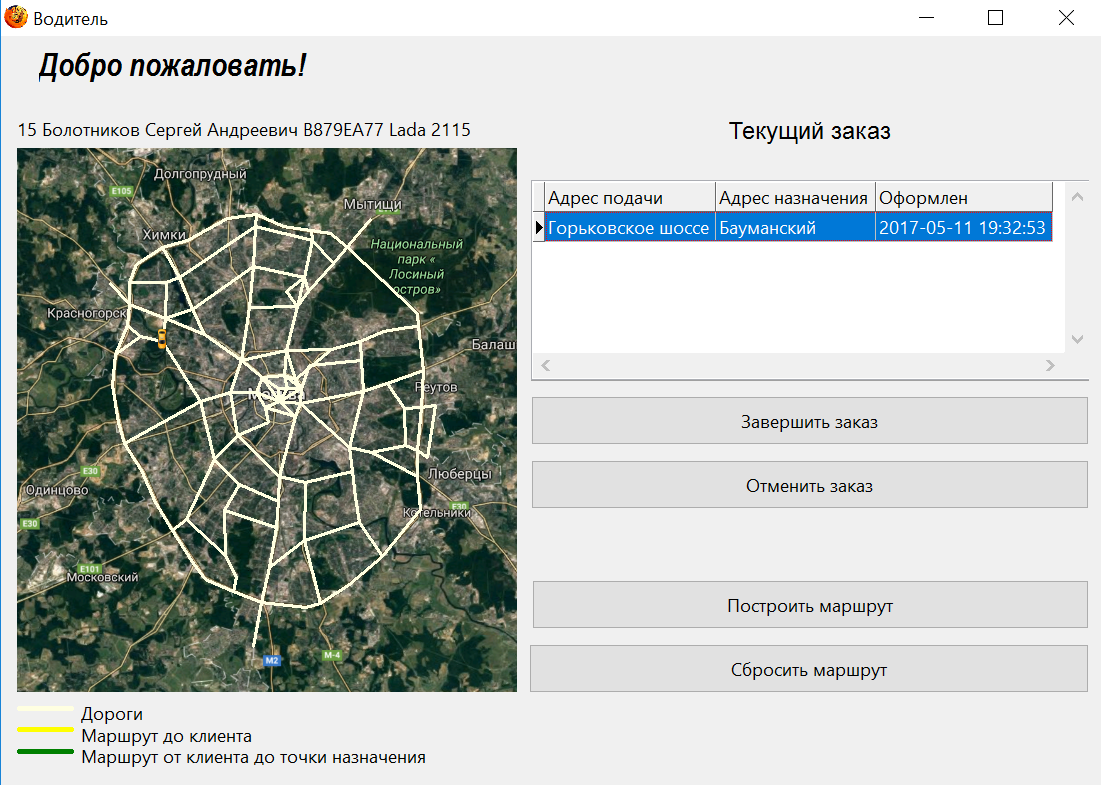


рис. 7. Сообщение «Неверный код доступа»

При нажатии на кнопку «Пассажир» пользователь (клиент) сможет беспрепятственно перейти в режим «Пассажир» (п. 5.4) для заказа такси.

**5.2. Режим «Водитель»**



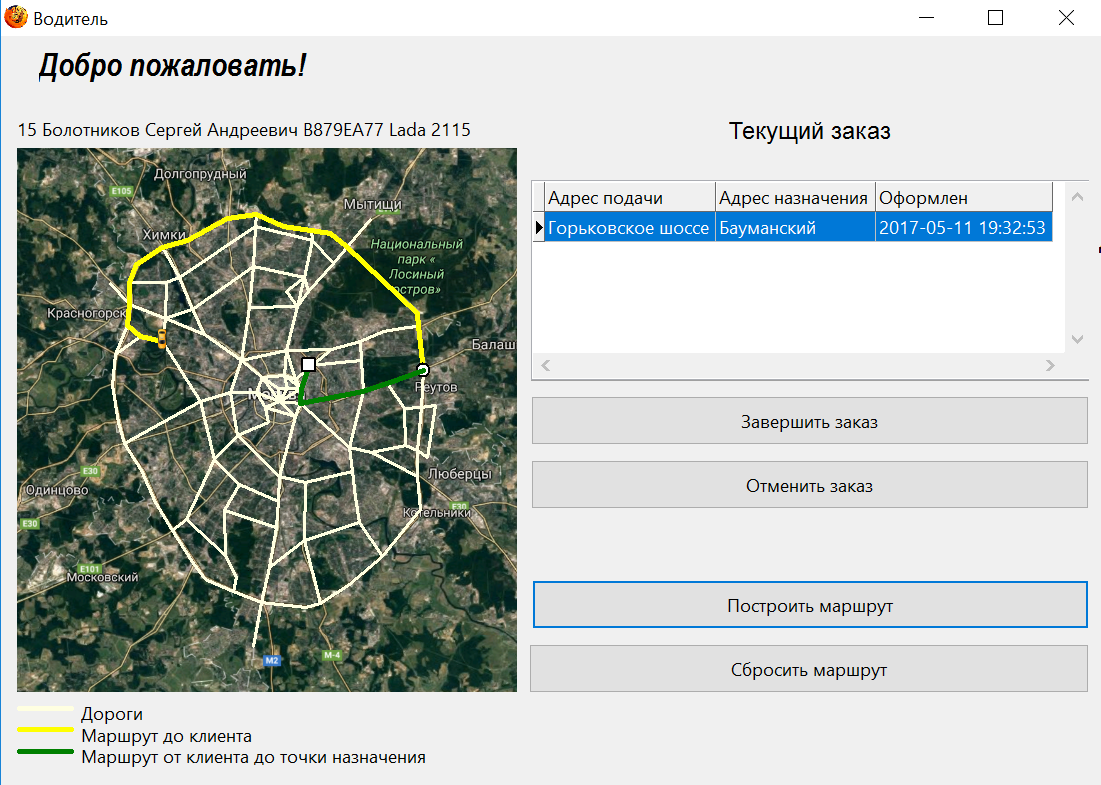
В данном режиме (рис. 8) пользователь (водитель) получает возможность управлять своими заказами.

рис. 8. Режим «Водитель»

В левой части экрана расположена карта города, на которой отображается местонахождение водителя в виде изображения желтого автомобиля.

Над картой программа выводит информацию о водителе и его транспортном средстве (ID, ФИО, автомобиль, госномер).

В правом верхнем углу окна отображается текущий заказ водителя, если такой имеется. Для того чтобы построить маршрут текущего заказа, водителю нужно нажать кнопку «Построить маршрут», после чего кратчайший маршрут от водителя до клиента отобразится на карте желтым цветом, а от клиента до точки назначения – зеленым (рис. 9).

рис. 9. Визуализация маршрута

Чтобы очистить карту, необходимо нажать кнопку «Сбросить маршрут».

Как только водитель довезет клиента до места назначения, ему следует нажать кнопку «Завершить заказ». После этого заказ исчезнет из списка текущих, получив статус «завершен», желтый автомобиль на карте переместится в конечный пункт заказа (таким образом, местоположение водителя обновится), а статус водителя изменится на «доступен».

Для отмены заказа водителю достаточно нажать кнопку «Отменить заказ», после чего заказ исчезнет из списка текущих, а статус водителя изменится на «доступен».

**5.3. Режим «Диспетчер»**

В данном режиме пользователь (диспетчер) имеет возможность просмотреть информацию, хранящуюся в СУБД MySQL.

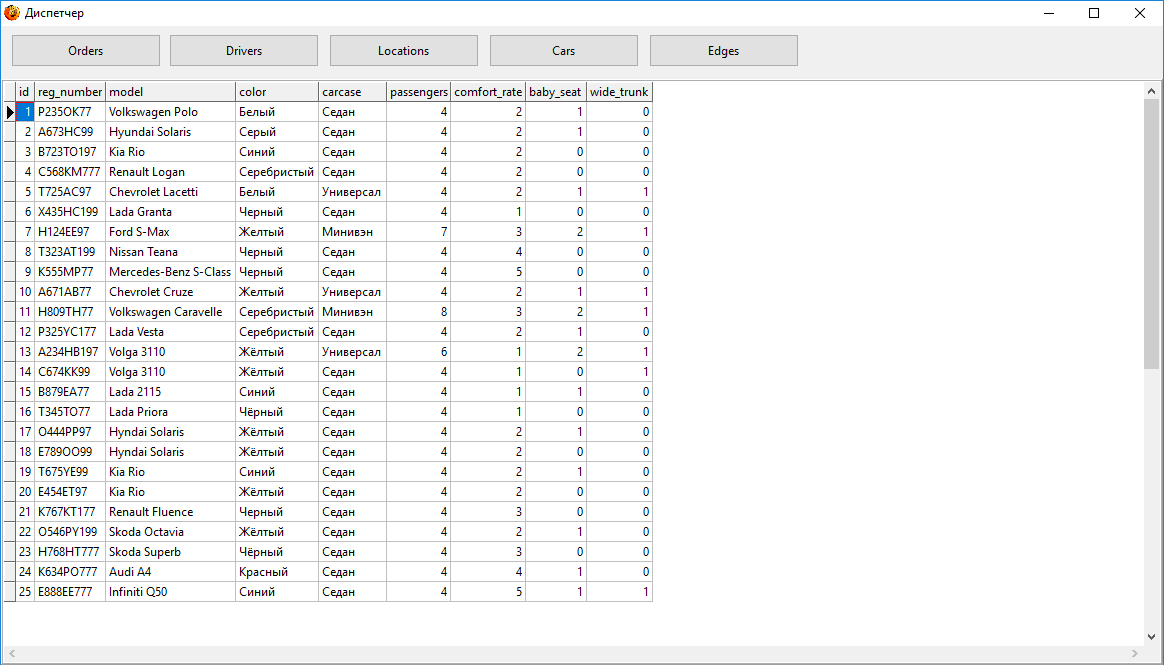
 Навигационная панель вверху окна (рис. 10) состоит из пяти кнопок и используется для переключения между различными таблицами (orders, drivers, locations, cars, edges). После нажатия на одну из пяти кнопок соответствующая таблица отобразится на экране.

рис. 10. Режим «Диспетчер»

**5.4. Режим «Пассажир»**

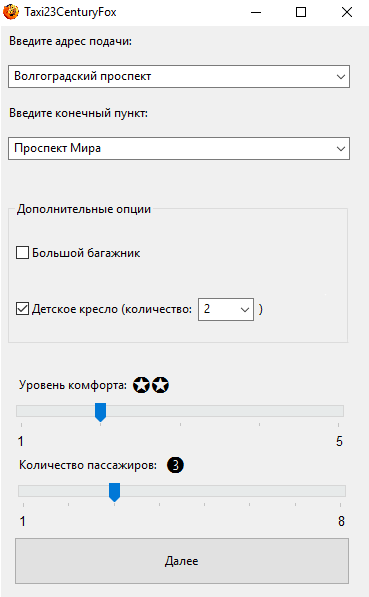
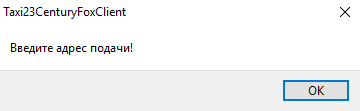
В данном режиме пользователь (заказчик) может заказать такси, выбрав на экране (рис. 11) необходимые параметры заказа.

рис. 11. Режим «Пассажир»

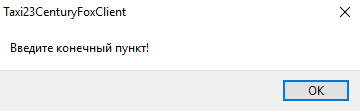
С помощью ниспадающих списков клиент выбирает адрес подачи такси и конечный пункт назначения среди предложенных адресов, взятых из базы данных. По желанию заказчик может выбрать дополнительные опции (автомобиль с большим багажником, детские кресла в количестве от 1 до 3 штук) с помощью компонентов CheckBox. По своему усмотрению пользователь выбирает уровень комфорта такси по шкале от 1 (низкий) до 5 (высокий) и количество пассажиров от 1 до 8. Когда все параметры введены клиенту достаточно нажать кнопку «Далее».

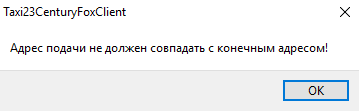
Далее приведены различные случаи, когда программа не даст оформить заказ и выведет соответствующее сообщение об ошибке.

при DBLookupComboBox1.Text = ‘ ’



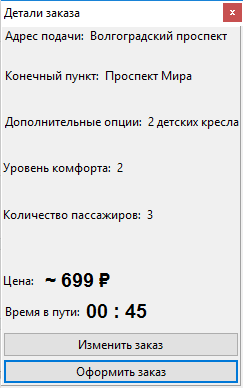
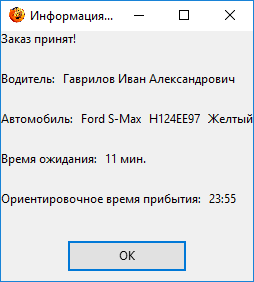
при DBLookupComboBox2.Text = ‘ ’



при DBLookupComboBox1.Text = DBLookupComboBox2.Text

Если все параметры выбраны верно, то программа выведет на экран окно для подтверждения заказа (рис. 12) с выбранными параметрами, рассчитанной стоимостью заказа и приблизительным временем в пути до точки назначения.

Если клиента что-то не устраивает, то он может изменить параметры заказа, щелкнув по кнопке «Изменить заказ». При этом произойдет возвращение к предыдущему окну (рис. 11).

 рис. 12. Детали заказа рис. 13. Информация о заказе

Для оформления заказа необходимо нажать кнопку «Оформить заказ». После чего программа подберет подходящего клиенту водителя (если таковой имеется) и выведет информацию о нем в новом окне (рис. 13). Здесь также отображаются время ожидания водителя и ориентировочное время прибытия в пункт назначения.

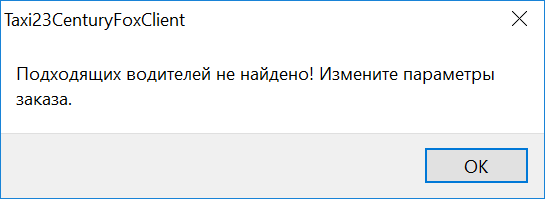


рис. 14. Сообщение «Водитель не найден»

Если доступных водителей, соответствующих выбранным параметрам, на данный момент нет, то на экран выводится сообщение с предложением изменить детали заказа (рис. 14), после чего происходит возвращение к окну с выбором параметров (рис. 11).

# **6. МЕТОД РЕШЕНИЯ ЗАДАЧИ**

## **6.1. Построение кратчайшего маршрута**

Построение кратчайшего пути происходит в два этапа: от водителя к пассажиру, затем от пассажира до точки назначения.

Для расчета каждого маршрута используется функция с алгоритмом Дейкстры, которая рассчитывает минимальное расстояние от исходной точки (передается при вызове функции) до каждой из точек графа. В данном случае функция возвращает массив, где индекс – номер вершины графа, а значение по этому индексу – номер вершины, из которой следует добираться в нее при построении кратчайшего пути.

Далее этот массив используется для восстановления маршрута (последовательности точек от конечного пункта к начальному). Это происходит путем перехода по индексу, совпадающему со значением в массиве, начиная с индекса конечного пункта до тех пор, пока не будет достигнут начальный пункт. Точки, проходимые при таком обходе, записываются в новый массив.

## **6.2. Подбор водителя**

Подбор водителя осуществляется с помощью хранимой процедуры в базе данных. Для этого используются следующие параметры:

- количество пассажиров (от 1 до 8)

- количество детских кресел (от 0 до 3)

- наличие большого багажника (есть/нет)

- уровень комфорта (от 1 до 5)

- расстояние до водителя (в минутах)

Все перечисленные параметры кроме последнего выбираются пользователем (пассажиром) при формировании заказа. Вычисление расстояния до всех водителей от данного пассажира происходит в отдельной подпрограмме, используя алгоритм Дейкстры.

С учетом параметров создается выборка подходящих водителей из числа доступных на данный момент; водители сортируются по возрастанию расстояния до них. Первому в этом списке водителю назначается данный заказ.

В случае успешного подбора (найден подходящий водитель) пассажиру выводится следующая информация:

- ФИО подобранного водителя

- модель, номер и цвет автомобиля

- время ожидания подачи машины

- ориентировочное время прибытия

Если водителя подобрать не удалось, пассажиру выводится соответствующее сообщение.

# **7. ОПИСАНИЕ АЛГОРИТМА**

**7.1. Структура данных**

Таблицы базы данных:

* cars – информация об автомобилях (ID, госномер, модель, цвет, кузов, максимальное кол-во пассажиров, уровень комфорта, кол-во детских кресел, объем багажника);
* drivers – информация о водителях (ID, ФИО, возраст, стаж, свободен / занят, автомобиль, местонахождение);
* edges – связи между вершинами графа (ID, первая вершина, вторая вершина, расстояние в минутах);
* locations – вершины графа (ID, название, координаты x и y);
* orders – информация о заказах (ID, время и дата оформления заказа, адрес подачи, конечный пункт, требуемые уровень комфорта и кол-во пассажиров, наличие большого багажника и (или) детского кресло (кресел), назначенный водитель, заказ завершен / нет).

**7.2. Выделение подпрограмм**

**7.2.1. Модуль Client.pas**

* procedure TfrmClient.btnConfirmClick (Sender: TObject) – проверка правильности ввода адресов.

**7.2.2. Модуль Details.pas**

* function Price (start, finish, comfort: integer): string – расчет стоимости заказа;
* function CountTime (start, finish: integer): integer – расчет времени в пути.

**7.2.3. Модуль Order\_Info.pas**

* function ConcatDriverDist (start: integer): string – склейка дистанций от пассажира до всех водителей в строку;
* procedure TfrmOrderInfo.FormShow (Sender: TObject) – вызов процедуры подбора водителя и вывод результата на экран.

**7.2.4. Модуль Driver\_Login.pas**

* procedure TfrmDriverLogin.OKClick (Sender: TObject) – проверка правильности введенного пароля.

**7.2.5. Модуль Driver.pas**

* procedure BuildRoads ( ) – построение дорог, отображение легенды карты;
* procedure DrawRoute (var path: nodesMass; routeColor: string) – визуализация маршрута на экране;
* procedure TfrmDriver.btnBuildRouteClick (Sender: TObject) – построение маршрута;
* procedure TfrmDriver.btnResetRoutesClick (Sender: TObject) – очистка карты от маршрутов;
* procedure TfrmDriver.btnCancelOrderClick (Sender: TObject) – отмена заказа;
* procedure TfrmDriver.btnFinishOrderClick (Sender: TObject) – процедура, позволяющая завершить заказ.

**7.2.6. Модуль Dispatch\_Login.pas**

* procedure TfrmDispatchLogin.OKClick (Sender: TObject) – проверка правильности введенного кода доступа.

**7.2.7. Модуль Dispatch.pas**

* procedure TfrmDispatch.CarsClick (Sender: TObject) – вывод таблицы cars на экран;
* Процедуры для вывода таблиц drivers, edges, locations и orders аналогичны предыдущей процедуре.

**7.2.8. Модуль Graph.pas**

* procedure FormAdjecencyMatrix (nodesCount: integer; var a: matrix) – формирование матрицы смежности;
* function ExtractMin (d: mass; var Q: massDyn): integer – поиск минимального значения расстояния в массиве вершин;
* function Dijkstra (var s: integer; param: string): mass – расчет минимального расстояния от исходной точки до каждой из вершин графа;
* function PathFromTo (start, finish: integer): nodesMass – формирование последовательности точек маршрута и их координат.

**7.2.9. SQL процедуры**

* PROCEDURE `all\_roads`() – получение координат вершин всех отрезков на карте (дорог);
* PROCEDURE `create\_order`(created datetime, start\_id int, finish\_id int, comfort\_rate int, passengers int, wide\_trunk int, baby\_seat int) – занесение деталей заказа в соответствующую таблицу;
* PROCEDURE `find\_matching\_driver`(distance\_array varchar(500), order\_id int) – подбор водителя по критериям заказа
* PROCEDURE `get\_drivers\_distance\_data`(distance\_array varchar(500)) – расформировывание строки с дистанциями до водителей (вспомогательная для `find\_matching\_driver`);
* PROCEDURE `finish\_order`(order\_id int) – обновление информации в базе данных после завершения заказа.

**7.3. Алгоритмы главной программы и подпрограмм**

**7.3.1. Модуль Client.pas**

procedure TfrmClient.btnConfirmClick (Sender: TObject)

входные параметры: –

выходные параметры: – или сообщение

нач

если DBLookupComboBox1.Text = ‘ ’ то

вывод («Введите адрес подачи!»)

иначе если DBLookupComboBox2.Text = ‘ ’ то

вывод («Введите конечный пункт!)

всё

иначе если DBLookupComboBox1.Text = DBLookupComboBox2.Text

вывод («Адрес подачи не может совпадать с конечным адресом!»)

всё

иначе

frmClient.Hide;

frmDetails.Show;

всё

кон

**7.3.2. Модуль Details.pas**

function Price (start, finish, comfort: integer): string

входные параметры:

* start (цел) – начальный пункт маршрута;
* finish (цел) – конечный пункт маршрута;
* comfort (цел) – уровень комфорта, выбранный пассажиром;

выходные параметры:

* price (цел) – рассчитанная стоимость заказа;

нач

если comfort = 1 то

money = 59

иначе если comfort = 2 или comfort = 3 то

money = 99

всё

иначе

money = 149

всё

dist = Dijkstra (start, ‘distance’) [finish]

если dist > 5 то

выбор (comfort)

comfort = 1: money = money + 10 \* (dist – 5)

comfort = 2: money = money + 15 \* (dist – 5)

comfort = 3: money = money + 20 \* (dist – 5)

comfort = 4: money = money + 25 \* (dist – 5)

comfort = 5: money = money + 30 \* (dist – 5)

квыбор

всё

Price = IntToStr (money)

кон

function CountTime (start, finish: integer): integer

входные параметры:

* start – начальный пункт маршрута;
* finish – конечный пункт маршрута;

выходные параметры:

* CountTime – рассчитанное время в пути;

нач

d = Dijkstra (start, ‘distance’)

dist = d[finish]

CountTime = dist

кон

**7.3.3. Модуль Order\_Info.pas**

function ConcatDriverDist (start: integer): string

входные параметры: start (цел) – точка, в которой находится пассажир;

выходные параметры: ConcatDriverDist (стр) – строка с дистанциями до водителей;

нач

map = Dijkstra (start, ‘distance’)

DriverLocationsQuery:

SQL = ‘select id, location from drivers order by id;’

цикл

ConcatDriverDist = ConcatDriverDist + IntToStr (map [FieldByName (‘location’).AsInteger]) + ‘,’

до EOF

кон

procedure TfrmOrderInfo.FormShow (Sender: TObject)

входные параметры: –

выходные параметры: –

нач

start = frmClient.DBLookupcomboBox1.KeyValue

FindMatchQuery:

SQL = ‘call find\_matching\_driver(:distance\_array, :order\_id);’

ParamByName('distance\_array').AsString := ConcatDriverDist(start);

ParamByName('order\_id').AsInteger := order\_id;

если EOF то

вывод («Подходящих водителей не найдено! Измените параметры заказа»)

иначе

{выходные параметры хранимой процедуры SQL}

вывод (name, model, reg\_number, color, dist)

всё

кон

**7.3.4. Модуль Driver\_Login.pas**

procedure TfrmDriverLogin.OKClick (Sender: TObject)

входные параметры: –

выходные параметры: –

если Password.Caption = ‘miem’ то

frmDriver.Show

frmDriverLogin.Close

frmStart.Hide

иначе

вывод («Неверный пароль»)

всё

**7.3.5. Модуль Driver.pas**

procedure BuildRoads ( )

входные параметры: –

выходные параметры: –

нач

DataModule1.RoadsQuery:

SQL = ‘call all\_roads;’

цикл

x1 := FieldByName('x1').AsInteger;

x2 := FieldByName('x2').AsInteger;

y1 := FieldByName('y1').AsInteger;

y2 := FieldByName('y2').AsInteger;

frmDriver.Map.Canvas.Pen.Color := clInfoBk;

frmDriver.Map.Canvas.Pen.Width := 3;

frmDriver.Map.Canvas.Line(x1, y1, x2, y2);

Next;

до EOF

кон

procedure DrawRoute (var path: nodesMass; routeColor: string)

входные параметры:

* path (nodesMass) – динамический массив точек в порядке прохождения их маршрутом от конца к началу. Каждая точка – структура с полями:
  + id – идентификатор вершины;
  + x – первая координата вершины;
  + y – вторая координата вершины;
* routeColor (стр) – цвет маршрута

выходные параметры: –

нач

цикл от i = 0 до length (path) – 2

если routeColor = ‘Yellow’ то

Canvas.Pen.Color = clYellow

иначе

Canvas.Pen.Color = clGreen

Canvas.Pen.Width = 5

Canvas.Line (path[i].x, path[i].y, path[i+1].x, path[i+1].y)

всё

кц

кон

procedure TfrmDriver.btnBuildRouteClick (Sender: TObject)

входные параметры: –

выходные параметры: –

нач

BuildRoads

SelectedOrders.DataSource.DataSet:

passenger\_location = FieldByName ('start\_id').AsInteger

destination = FieldByName ('finish\_id').AsInteger

если driver\_location.id <> passenger\_location то

path = PathFromTo (driver\_location.id, passenger\_location)

DrawRoute (path, 'Yellow')

всё

path = PathFromTo (passenger\_location, destination)

DrawRoute (path, 'Green')

кон

procedure TfrmDriver.btnResetRoutesClick (Sender: TObject)

входные параметры: –

выходные параметры: –

нач

Map.Repaint

BuildRoads

Car.Repaint

кон

TfrmDriver.btnCancelOrderClick (Sender: TObject)

входные параметры: –

выходные параметры: –

нач

CancelOrderQuery:

SQL = ‘update orders set driver = NULL where id = ' + IntToStr(SelectedOrders.DataSource.DataSet.FieldByName('id').Value) + ';’

вывод («Заказ отменен»)

кон

procedure TfrmDriver.btnFinishOrderClick (Sender: TObject)

входные параметры: –

выходные параметры: –

нач

FinishOrderQuery:

ParamByName('order\_id').AsInteger =

SelectedOrders.DataSource.DataSet. FieldByName ('id').Value

вывод («Заказ завершен»)

driver\_location:

id := FieldByName('id').AsInteger

x := FieldByName('x').AsInteger

y := FieldByName('y').AsInteger

Car.Left := x-4

Car.Top := y-28

btnResetRoutes.Click

кон

**7.3.6. Модуль Dispatch\_Login.pas**

procedure TfrmDispatchLogin.OKClick (Sender: TObject)

входные параметры: –

выходные параметры: –

если Password.Caption = ‘2323’ то

frmDispatch.Show

frmDispatchLogin.Close

frmStart.Hide

иначе

вывод («Неверный код доступа»)

всё

**7.3.7. Модуль Dispatch.pas**

procedure TfrmDispatch.CarsClick (Sender: TObject)

входные параметры: –

выходные параметры: –

нач

SQLQuery1:

SQL = ‘select \* from cars;’

кон

Процедуры для таблиц drivers, edges, locations и orders аналогичны процедуре TfrmDispatch.CarsClick.

**7.3.8. Модуль Graph.pas**

procedure FormAdjecencyMatrix (nodesCount: integer; var a: matrix)

входные параметры:

* nodesCount (цел) – количество вершин графа

выходные параметры:

* a – матрица смежности

нач

inf = 9999

//инициализация всех возможных ребер

цикл от i = 1 до nodesCount

цикл от j = 1 до nodesCount

a[i][j] = inf

//заполнение матрицы существующими ребрами

DataModule1.EdgesQuery:

цикл-пока не (EOF)

n = FieldByName(‘first\_node’).AsInteger

m = FieldByName(‘second\_node’).AsInteger

a[n][m] = FieldByName(‘distance’).AsInteger

a[m][n] = FieldByName(‘distance’).AsInteger

Next

кц

кон

function ExtractMin (var d: mass; var Q: massDyn): integer;

входные параметры:

* d (mass[1:100]) – массив целых чисел, индексы которого соответствуют идентификаторам вершин, а значения – дистанции до каждой из них (в минутах);
* Q - динамический массив, где хранятся все вершины, наименьшее расстояние до которых еще не известно;

выходные параметры:

* index (цел) – идентификатор вершины, расстояние до которой от исходной точки является наименьшим из всех точек в Q;

нач

//поиск элемента с минимальным расстоянием

inf = 9999

min = inf

цикл от i = 0 до length(Q) – 1

если d[Q[i]] < min то

min = d[Q[i]]

index = Q[i]

всё

кц

//удаление найденного элемента из массива Q

i = 0

цикл-пока i < length(Q) и Q[i] ≠ index

i = i + 1

кц

если i < length (Q) то

Q[i] = Q[length(Q) - 1]

setlength(Q, length(Q) - 1)

всё

ExtractMin = index

кон

function Dijkstra (s: integer; param: string): mass;

входные параметры:

* s (цел) – идентификатор вершины, от которой следует вести отсчет
* param (стр) – параметр, значением которого определяется возвращаемый результат

выходные параметры:

* d (mass[1:100]) – массив целых чисел, индексы которого соответствуют идентификаторам вершин, а значения – дистанции до каждой из них (в минутах)

ИЛИ

* p (mass[1:100]) – массив целых чисел, где индекс – номер вершины графа, а значение по этому индексу – номер вершины, из которой следует добираться в нее при построении кратчайшего пути

нач

inf = 9999

//получение количества вершин графа

DataModule1.CountLocationsQuery:

SQL = ‘select count(\*) as nodesCount from locations’

nodesCount = FieldByName(‘nodesCount’).AsInteger

//создание и заполнение матрицы смежности a

setlength(a, nodesCount + 1) // 0-я строка и столбец не будут учитываться

цикл от i = 1 до nodesCount

setlength (a[i], nodesCount + 1)

кц

{пересечения строка-столбец, где есть дорога, заполняются значением дистанции}

FormAdjecencyMatrix (nodesCount, a)

{изначально дистанция до всех точек принимается за бесконечность,

а для исходной точки – ноль.}

цикл от i = 1 до nodesCount

d[i] = inf

кц

d[s] = 0

//предыдущих точек маршрута изначально нет – заполняем массив нулями

цикл от i = 1 до nodesCount

p[i] = 0

кц

{Заполнение динамического массива, где будут храниться все вершины, наименьшее расстояние до которых еще не известно}

цикл от i = 1 до nodesCount

setlength(q, length(q)+1)

q[i - 1] = i;

кц

цикл-пока length(q) > 0

{извлечение из очереди вершины с наименьшей длиной - расстояние

до нее теперь точно известно}

curr = ExtractMin(d, Q)

{обновление массивов с учетом нового точно оптимального маршрута до curr }

цикл от i = 1 до nodesCount

если d[i] > d[curr] + a[curr][i] то

d[i] = d[curr] + a[curr][i]

p[i] = curr

всё

кц

кц

если param = ‘distance’ то

Dijkstra = d

иначе если param = ‘previous’ то

Dijkstra = p

всё

кон

function PathFromTo (start, finish: integer): nodesMass;

входные параметры:

* start (цел) – начальный пункт
* finish (цел) – конечный пункт

выходные параметры:

* path (nodesMass) – динамический массив точек в порядке прохождения их маршрутом от конца к началу. Каждая точка – структура с полями:
  + id – идентификатор вершины
  + x – первая координата вершины
  + y – вторая координата вершины

нач

//получение массива p из предыдущей функции

prevNodes = Dijkstra (start, ‘previous’)

//Восстановление маршрута в поле id массива path в обратном порядке

next = finish

цикл

setlength(path, length(path) + 1)

path[length(path) - 1].id = next

next = prevNodes[next]

до next = start

setlength(path, length(path) + 1)

path[length(path) - 1].id = start

//Получение координат каждого промежуточного пункта

цикл от i = 0 до length(path) – 1

DataModule1.NodeCoordinatesQuery:

SQL = ‘select x, y from locations where id = :id’

ParamByName('id').AsInteger = path[i].id

path[i].x = FieldByName('x').AsInteger

path[i].y = FieldByName('y').AsInteger

кц

PathFromTo = path

кон

**7.3.9. SQL-процедуры**

PROCEDURE `all\_roads`()

входные параметры: –

выходные параметры:

* x1, y1 – координаты начальной вершины
* x2, y2 – координаты конечной вершины

Алгоритм:

1. Объединить таблицы locations и edges как t1, используя поля edges.first\_node = locations.id;
2. Выбрать в t1 поля: locations.x как x1, locations.y как y1, edges.id\_edge как id;
3. Объединить таблицы locations и edges как t2, используя поля edges.second\_node = locations.id;
4. Выбрать в t2 поля: locations.x как x2, locations.y как y2, edges.id\_edge как id
5. Объединить t1 и t2, используя t1.id = t2.id;
6. Выбрать x1, x2, y1, y2 и упорядочить по возрастанию id.

PROCEDURE `create\_order`(created datetime, start\_id int, finish\_id int,

comfort\_rate int, passengers int, wide\_trunk int, baby\_seat int)

входные параметры:

* created (дата и время) – время и дата создания заказа
* start\_id (цел) – идентификатор адреса подачи
* finish\_id (цел) – идентификатор адреса назначения
* comfort\_rate (цел) – уровень комфорта
* passengers (цел) – количество пассажиров
* wide\_trunk (цел) – наличие большого багажника
* baby\_seat (цел) – количество детских кресел

выходные параметры:

* id (цел) – номер заказа

Алгоритм:

1. Вставить в поля таблицы orders (created, start\_id, finish\_id, comfort\_rate, passengers, wide\_trunk, baby\_seat) соответствующие значения параметров;
2. Выбрать поле id из таблицы orders, упорядочив по убыванию, ограничить одной строкой.

PROCEDURE `find\_matching\_driver`(distance\_array varchar(500), order\_id int)

входные параметры:

* distance\_array (стр) – строка, в которой через запятую перечислены значения дистанции от клиента, сделавшего заказ, до каждого из водителей (значения упорядочены в порядке возрастания id водителя)
* order\_id (цел) – номер заказа, для которого следует подобрать водителя

выходные параметры:

* id (цел) – номер подобранного водителя
* dist (цел) – дистанция в минутах от местоположения водителя до адреса подачи такси
* name (стр) – ФИО водителя
* model (стр) – модель автомобиля
* color (стр) – цвет автомобиля
* reg\_number (стр) – госномер автомобиля

Алгоритм:

1. Создать временную таблицу driver\_search с полями id, available, dist, comfort\_rate, passengers, baby\_seat, wide\_trunk, name, model, color, reg\_number;
2. Вызвать процедуру get\_drivers\_distance\_data (distance\_array) для преобразования строки distance\_array в таблицу driver\_dist;
3. Вставить данные из таблиц driver\_dist, driver, cars в таблицу driver\_search:
   1. Объединить таблицы drivers, cars и driver\_dist, используя drivers.car\_id = cars.id и drivers.id = driver\_dist.id;
   2. Выбрать для вставки поля drivers.id как id, available, dist, comfort\_rate, passengers, baby\_seat, wide\_trunk, name, model, color, reg\_number;
4. Вывести данные подходящего водителя:
   1. Объединить таблицы driver\_search и orders
   2. Выбрать поля driver\_search.id, dist, name, model, color, reg\_number, где:

order.id = order\_id

driver\_search.available = 1

driver\_search.passengers >= order.passengers

driver\_search.baby\_seat >= order.baby\_seat

driver\_search.wide\_trunk >= order.wide\_trunk

| driver\_search.comfort\_rate – order.comfort\_rate | <= 1

* 1. Упорядочить по возрастанию dist;
  2. Ограничить одной строкой;

1. Присвоить driver\_id = driver\_search.id (первая строка);
2. Если driver\_id не NULL, то:
   1. Обновить таблицу drivers: поле available = 0, где id = driver\_id;
   2. Обновить таблицу orders: поле driver = driver\_id, где id = order\_id;

PROCEDURE `get\_drivers\_distance\_data`(distance\_array varchar(500))

входные параметры:

* distance\_array (стр) – строка, в которой через запятую перечислены значения дистанции от клиента, сделавшего заказ, до каждого из водителей (значения упорядочены в порядке возрастания id водителя)

выходные параметры:

* driver\_dist – таблица с полями:
  + id (цел) – номер водителя
  + dist (цел) – дистанция от клиента до местоположения водителя в минутах

Алгоритм:

1. Создать временную таблицу driver\_dist с полями id, dist;
2. Выбрать значения id из таблицы drivers, упорядочив по возрастанию id;
3. Вставить значения п.2 в таблицу driver\_dist (поле id)
4. Разбить строку distance\_array на отдельные числа:
   1. Присвоить i = 1;
   2. Посчитать количество id в таблице driver\_dist и присвоить это значение count;
   3. цикл-пока i <= count

-- найти позицию первой запятой

coma\_position = locate(',', distance\_array)

-- копировать все до запятой

dist\_str = substring(distance\_array from 1 for coma\_position-1);

-- преобразовать в число и занести в первую строку с пустым

значением поля dist

Обновить таблицу driver\_dist:

поле dist = cast(dist\_str as unsigned),

где dist = NULL,

упорядочить по возрастанию id,

ограничить первой строкой;

-- оставить в строке все, что после запятой

distance\_array = substring(distance\_array from

coma\_position+1);

i = i + 1;

до i > count;

1. Выбрать все поля из driver\_dist.

PROCEDURE `finish\_order`(order\_id int)

входные параметры:

* order\_id (цел) – номер заказа, который следует завершить

выходные параметры:

* id (цел) – номер вершины, являющейся пунктом назначения
* x, y (цел) – координаты пункта назначения

Алгоритм:

1. Обновить таблицу orders: поле completed = 1, где id = order\_id;
2. Выбрать из таблицы orders поле driver, где id = order\_id и присвоить это значение переменной dr;
3. Выбрать из таблицы orders поле finish\_id, где id = order\_id и присвоить это значение переменной new\_location;
4. Обновить таблицу drivers: поля location = new\_location, available = 1,

где id = dr;

1. Выбрать из таблицы locations поля id, x, y, где id = new\_location.

# **8. СТРУКТУРА ПРОГРАММЫ**

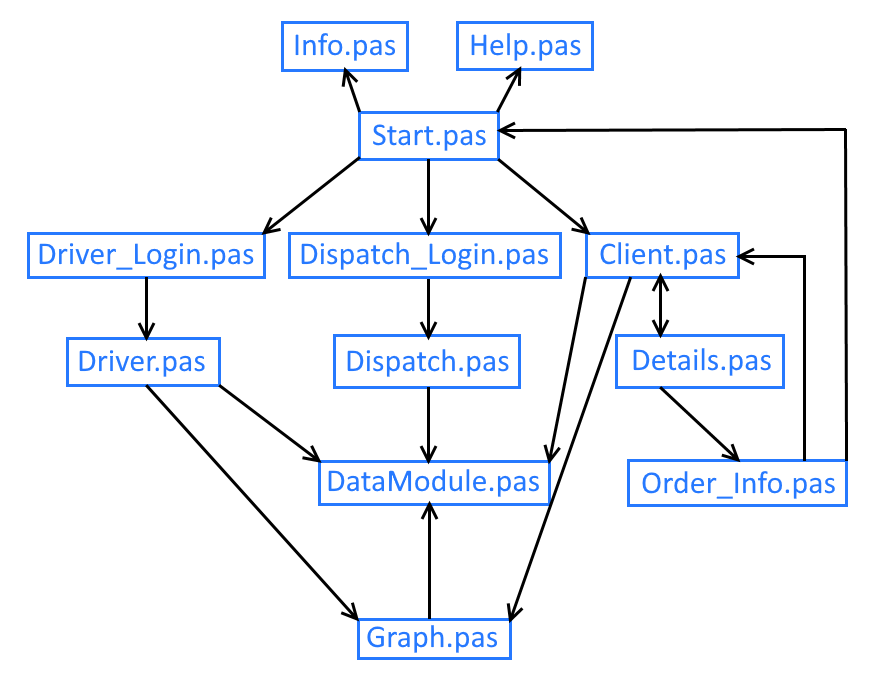


рис. 15. Схема связей

**10. РАСПЕЧАТКА ТЕСТОВ И РЕЗУЛЬТАТОВ РАБОТЫ ПРОГРАММЫ**

**10.1. Тест №1**

|  |  |
| --- | --- |
| Входные параметры | Результат работы программы |
|  |  |

**10.2. Тест №2**

|  |  |
| --- | --- |
| Входные параметры | Результат работы программы |
|  |  |