МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЩОВАНИЯ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. Шухова»

(БГТУ им. В. Г. Шухова)

Курсовая работа

Дисциплина «Сети ЭВМ и телекоммуникации»

На тему: «Алгоритмы маршрутизации в сети»

Выполнил: студент группы ВТ-31

Ковалёв Илья

Проверил: Федотов Е. А.

г. Белгород

2020 г.

**Оглавление**

[1. Вступление 3](#_Toc41660342)

[2. Теоретические сведения 4](#_Toc41660343)

[2.1 Требования к алгоритмам маршрутизации 4](#_Toc41660344)

[2.2 Классификация алгоритмов маршрутизации 6](#_Toc41660345)

[2.3 Алгоритм затопления 8](#_Toc41660346)

[2.4 Дистанционно-векторный алгоритм 9](#_Toc41660347)

[2.5 Алгоритм с учетом состояния связи 10](#_Toc41660348)

[3. Разработка программы 12](#_Toc41660349)

[5. Руководство пользователя 20](#_Toc41660350)

[6. Заключение 21](#_Toc41660351)

[7. Список использованных источников 22](#_Toc41660352)

[8. Приложения 23](#_Toc41660353)

# **Вступление**

Маршрутизация представляет собой процесс определения маршрута следования информации в сетях связи. Алгоритмы маршрутизации применяются для определения наилучшего пути пакетов от источника к приемнику и являются основой любого протокола маршрутизации. Как правило, задача маршрутизации в сети рассматривается как графовая задача, в которых маршрутизаторы представляются как узлы графа, а физические линии, которые проложены между ними – ребрами графа.

Основной функцией маршрутизатора является чтение заголовков пакетов сетевых протоколов и принятие решения об дальнейшем маршруте следования пакета по сети. Задачами маршрутизатора являются:

1. Построение таблицы маршрутизации
2. Определение на основе таблицы маршрутизации маршрута
3. Буферизация, фрагментация и фильтрация поступающих пакетов

Функции маршрутизатора могут быть разбиты на 3 группы в соответствии с уровнями модели OSI[1]:

1. Уровень интерфейсов – обеспечивает физический интерфейс со средой передачи данных.
2. Уровень сетевых протоколов – анализирует содержимое заголовка сетевого уровня, фильтрует трафик.
3. Уровень протоколов маршрутизации – на основании этих протоколов маршрутизаторы обмениваются информацией о топологии сети и строят таблицы маршрутизации.

Далее рассмотрим основные цели и классификации алгоритмов маршрутизации, а также требования к ним.

**Цель работы:** реализовать один из алгоритмов маршрутизации в сетях. Изучить алгоритмы маршрутизации пакетов, провести исследование этих алгоритмов, выбрать один из исследуемых. Реализовать этот алгоритм в виде программы. Сделать выводы.

# **Теоретические сведения**

## **2.1 Требования к алгоритмам маршрутизации**

1. Оптимальность

Является основным требованием. Данный показатель алгоритма описывает его способность наиболее удобный маршрут, который напрямую зависит от показателей и от веса этих показателей, которые используются при проведении расчета оптимальности того или иного маршрута.

1. Простота

Алгоритм маршрутизации должен быть как можно более простым, то есть выполнять свои задачи и при этом затрачивать минимальное количество вычислительных ресурсов, чтобы работать на сравнительно маломощных ЭВМ.

1. Живучесть

Алгоритмы маршрутизации должны четко функционировать при любом состоянии сети, не завися от непредвиденных обстоятельств, таких как отказы аппаратуры, наличие высокой нагрузки и прочее. Так как маршрутизаторы расположены в узловых точках сети, их отказ вызовет серьезные проблемы.

1. Быстрая сходимость

Сходимость – процесс, в течении которого достигается соглашение между всеми маршрутизаторами по оптимальным маршрутам. В случае, если некоторое событие в сети приводит к тому, что маршрутизаторы либо становятся недоступными, либо же наоборот, маршрутизаторы рассылают сообщение об обновлении маршрутизации, после которого происходит пересчет оптимальных маршрутов. Алгоритм, который не позволит быстро пересчитать оптимальные маршруты в сети, нельзя назвать эффективным.

1. Гибкость

Алгоритмы маршрутизации должны быстро и точно адаптироваться к различным обстоятельствам в сети, например, в случае если сегмент сети был отвергнут, все маршруты, которые использовали этот сегмент, заменяются на следующие наилучшие маршруты. Некоторые алгоритмы могут быть реализованы таким образом, чтобы они могли адаптироваться к изменениям полосы пропускания сети, размеров очереди к маршрутизатору, величины задержки сети и других параметров.

## **2.2 Классификация алгоритмов маршрутизации**

По способу выбора наилучшего маршрута можно выделить такие группы:

* Одношаговые – каждый маршрутизатор при выборе маршрута определяет только одно звену формируемого пути.
* Многошаговые – весь маршрут задается уже в отправленном пакете узлом источника. Такие алгоритмы также называются алгоритмами маршрутизации от источника.

По способу построения таблицы маршрутизации:

* Простой маршрутизации – таблицы маршрутов, как правило, не составляются, либо же являются примитивными и не передают никакой маршрутной информации.

К таким можно отнести алгоритмы случайной маршрутизации (пакеты посылаются в случайном направлении), лавинная маршрутизация, или алгоритм затопления (пакеты посылаются во все выходные направления), алгоритмы кратчайшей очереди (пакеты идут на порт, который менее всего загружен), алгоритмы, базирующиеся на предыдущем опыте (присутствует примитивная таблица маршрутизации, которая дублирует предыдущую передачу).

* Статические – при статической маршрутизации таблицы маршрутов формируются при генерации сети и после, как правило, не изменяются. Таблица маршрутов редактируется вручную в случаях, при которых некоторый узел вышел из строя либо же узел был добавлен в сети.
* Адаптивные или динамические – данные алгоритмы обеспечивают автоматическое обновление таблиц при изменении конфигурации сети. Протоколы, которые основаны на основе динамических алгоритмов, позволяют маршрутизаторам собирать основную информацию о топологии связей в сети, обрабатывая все изменении в конфигурации связей. В таблицах маршрутизации обычно имеется значение, которое называется временем жизни маршрута - информации об некотором промежутке времени, в течении которого данный маршрут считается действительным.

Динамическая маршрутизация является сложным процессом, который включает в себя такие задачи:

Формирование маршрутов, осуществляемое с помощью алгоритмов маршрутизации путём составления в каждом узле коммутации таблиц маршрутов пакетов

* Реализацию маршрутов
* Контроль состояния сети
* Передачу информации о состоянии сети
* Корректировку маршрутов

По виду информации, которой обмениваются маршрутизаторы:

Дистанционно-векторные алгоритмы – каждый маршрутизатор периодически передает всем соседним маршрутизаторам вектор сообщения, которое содержит адреса всех известных ему подсетей и расстояние до них. При этом каждому маршруту присваивается время жизни, которое автоматически декрементируется. Если это значение не было обновлено за выделенное время – маршрут считается недействительным. Главным недостатком является то, что информация получается с помощью соседних маршрутизаторов, а не напрямую, что создает дополнительную нагрузку на сеть.

Алгоритмы состояния связи – каждый маршрутизатор обеспечивается всей необходимой для построения точной топологией сети информацией. В данном случае каждый маршрутизатор формирует сообщение, которое рассылается всем соседним маршрутизатором, и содержит информацию о адресе соседней подсети, типе интерфейса и метрику интерфейса, т. е. пропускные способности каждого из путей, время задержки и некоторый коэффициент надежности. Соседний же маршрутизатор, получая эту информацию, делает аналогичные действия, при этом обрабатывает и записывает полученные данные в таблицу маршрутизации.

Данный тип алгоритмов является наиболее распространенным, поскольку гарантирует надежную работу и масштабируемость, что и отличает его от дистанционно-векторных алгоритмов, которые эффективно функционируют в простых сетях небольшого размера.

Рассмотрим подробнее некоторые из вышеперечисленных алгоритмов и протоколы, в которых они используются.

## **2.3 Алгоритм затопления**

Алгоритм затопления (flooding) является одним из самых простых алгоритмов маршрутизации для распространения информации по сети. Данный алгоритм является неадаптивным и работает по такому принципу:

При получении пакета каждый узел пересылает его всем соседним узлам, исключая тот, от которого пришел пакет. Является неэффективным, особенно в случае, если он функционирует в крупной сети с большим количеством узлов, так как создает значительную нагрузку на сеть.

Еще одним недостатком данного алгоритма является то, что он не уведомляет отправителя об успешном получении пакета всеми узлами сообщения. Для устранения этого недостатка может использоваться улучшение этого алгоритма, которое получило название Flooding with Acknowledge – затопление с подтверждением. В этом случае каждый узел, получив пакет, отправляет его всем соседям, которые, в свою очередь, отправляют обратно сообщение об успешной доставке пакета, если он был успешно отправлен дальше.

Другим недостатком является избыточность пересылаемых данных в сетях с топологией, отличной от древовидной, но этот недостаток легко исправляется с помощью добавления каждому узлу некоторого буфера, который хранит идентификаторы уже пересланных пакетов, чтобы не отправлять их еще раз.

Среди преимуществ данного алгоритма можно отметить такие:

* Гарантированная доставка пакета
* Высокая эффективность при широковещательной рассылке
* Надежность
* Не требует дополнительной конфигурации, потому может использоваться как часть более сложного алгоритма
* Удобен для отладки, поскольку находит все пути от источника к получателю, среди которых, вероятнее всего, найдется кратчайший.

## **2.4 Дистанционно-векторный алгоритм**

Функционирование данного алгоритма подразумевает, что каждый маршрутизатор периодически широковещательно рассылает по сети вектор, компонентами которого являются расстояния (представляются некоторой метрикой) от данного маршрутизатора до всех известных ему узлов. Получив от соседнего узла вектор расстояний до известных тому сетей, маршрутизатор увеличивает компоненты вектора на величину расстояния от себя до отправителя вектора, и дополняет его информацией об известных ему других сетях. Обновленное значение вектора вновь рассылается соседям. В итоге каждый маршрутизатор благодаря информации от соседних маршрутизаторов будет знать информацию о топологии сети и расстояния до каждого из узлов. После этого выбирается наиболее эффективный маршрут, т. е. тот, который имеет самое низкое значение метрики. Маршрутизатор, который передал информацию о данном маршруте, помечается в таблице маршрутизации как следующий.

Данный алгоритм эффективно работает в небольших сетях, в больших же довольно часто линии связи забиваются служебным трафиком, что негативно сказывается на производительность самой сети.

Недостатком также является то, что маршрутизаторы не владеют достаточной информацией о топологии связей в сети, а располагают только косвенной информацией, потому смена конфигурации сети не всегда корректно обрабатывается данным алгоритмом.

Дистанционно-векторный алгоритм применяется в протоколе RIP (Routing Information Protocol)[2].

## **2.5** **Алгоритм с учетом состояния связи**

Данный алгоритм более сложен в реализации, поскольку устанавливает такие требования к маршрутизаторам:

* Маршрутизатор должен обнаруживать соседей, и узнавать расстояния до них
* Задавать метрику расстояния с каждым из своих соседей
* Создавать пакет, содержащий всю собранную информацию
* Посылать этот пакет другим маршрутизаторам и принимать от них аналогичные пакеты
* Вычислять кратчайший путь до каждого присутствующего маршрутизатора в сети

Рассмотрим этот алгоритм пошагово.

1. Определение соседних маршрутизаторов

Как только маршрутизатор появляется в сети, ему необходимо получить информацию от соседей. Для этого он посылает специальный пакет во все соседние узлы. В ответ он получает пакеты, которые содержат идентификаторы соседних узлов, которые, очевидно, должны быть уникальными.

1. Задание метрики расстояний

Метрика расстояния между двумя узлами, как правило, задается автоматически или оператором сети. Чаще всего она задается значением, обратно пропорциональным пропускной способности линии, благодаря чему будут выбираться пути с наименьшей задержкой.

1. Создание пакета состояния линий

На данном этапе формируется специальный пакет, который будет содержать идентификаторы соседей и расстояние между ними. Обычно, эти пакеты создаются через определенные интервалы времени, либо при получении сигнала об изменении в сети, например, сигнал о том, что некоторый маршрутизатор вышел из строя.

1. Распространение пакетов состояния линий

После формирования специального пакета состояния линий, этот пакет рассылается всем соседям для получения точной топологии связей сети. Такие пакеты являются приоритетными, и должны приниматься безотказно.

1. Вычисление новых маршрутов

Собрав полный комплект пакетов состояния линий, маршрутизатор может построить полный граф сети, учитывая полученные данные о состоянии линий. На данном этапе применяется алгоритм Дейкстры для поиска кратчайших путей.

Недостатками данного алгоритма являются необходимость наличия у маршрутизаторов некоторого количества памяти для хранения топологии сети и вычислительные мощности, которые необходимы для анализа полученных данных.

Преимуществами же можно назвать наличие точных данных у маршрутизаторов о топологии сети, что исключает возможность возникновения петель, которые могут привести к выходу из строя целого сегмента сети.

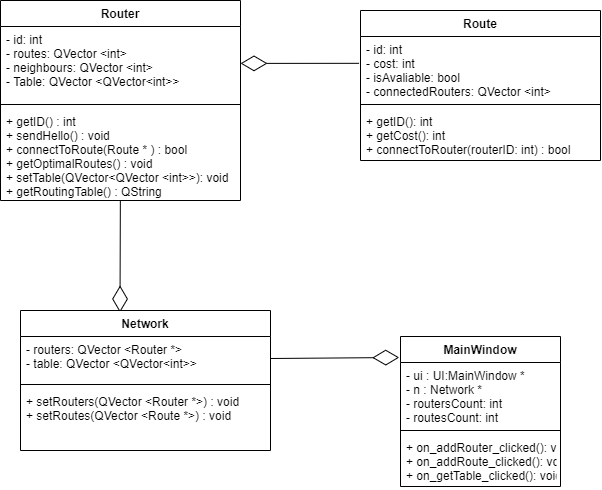
Далее будет реализован данный алгоритм.

# **Разработка программы**

Требования к разрабатываемой программе:

* Пользователь может ввести данные о существующих в сети маршрутизаторах и линиях связи.
* Каждый маршрутизатор после включения в сеть должен отправлять приветственное сообщение соседним маршрутизаторам, чтобы узнать их ID и метрику канала связи.
* После подключения все маршрутизаторы должны обменяться специальными сообщениями, которые будут содержать ID соседних маршрутизаторов и стоимость пути до них.
* По полученным данным каждый маршрутизатор строит минимальные пути до остальных маршрутизаторов в сети, после чего сформировать таблицу маршрутизации, которая будет выведена в разрабатываемом приложении.

Для правильного построения архитектуры разрабатываемого приложения создадим UML-диаграмму:



В процессе реализации программы были разработаны такие классы:

Класс Route – реализует линию связи. Содержит в себе такие данные:

* Id – идентификатор линии связи
* Cost – метрика канала связи. задается случайным числом, чтобы уменьшить количество вводимых пользователем данных в демонстрационной программе.
* isAvaliable – указывает на доступность канала связи в данный момент времени.
* ConnectedRouters – массив, содержащий ID подключенных роутеров. Необходим для валидации вводимых данных (В том числе, чтобы не были добавлены петли).

Класс Network, реализующий сеть в целом. Содержит массив созданных пользователем маршрутизаторов и линий связи. Был реализован метод void sendTable(), с помощью которого маршрутизаторы обменивались данными о существующих линиях связи и пр.

Листинг метода void sendTalbe():

void Network::**sendTable**(){

*for* (int i = 0; i < routers.size(); i++){

routers[i]->setTable(*this*->table);

}

}

Как видно, используется метод void setTable класса Router.

Класс Router, реализующий маршрутизатор, содержащий:

* Id – идентификатор маршрутизатора
* Routes – массив указателей на подключенные к маршрутизатору линии связи.
* Table – таблица, в которой изначально хранятся данные о соседних маршрутизаторах, после выполнения алгоритма Дейкстры хранит полную таблицу маршрутизации по всей сети конкретного роутера.

Для подключения маршрутизатора в сеть был создан метод bool connectToRoute(Route \*r), который подключал маршрутизатор каналу связи, передаваемый в качестве единственного аргумента.

Листинг метода bool connectToRoute:

bool Router::**connectToRoute**(Route \* r){

*if* (routes.contains(r)) *return* *false*;

routes.append(r);

r->connectToRouter(*this*->id);

*return* *true*;

}

В случае, если к маршрутизатору уже подключена линия связи, передаваемая в качестве аргумента, метод вернет значение false, если же подключение пройдет успешно – true.

Для отправки приветственного сообщения был создан метод void sendHello(), который отправлял приветственное сообщение соседним маршрутизаторам.

void Router::**sendHello**(){

*for* (int i = 0; i < routes.size(); i++){

neighbours.append(routes[i]->getNeighbour(*this*->id));

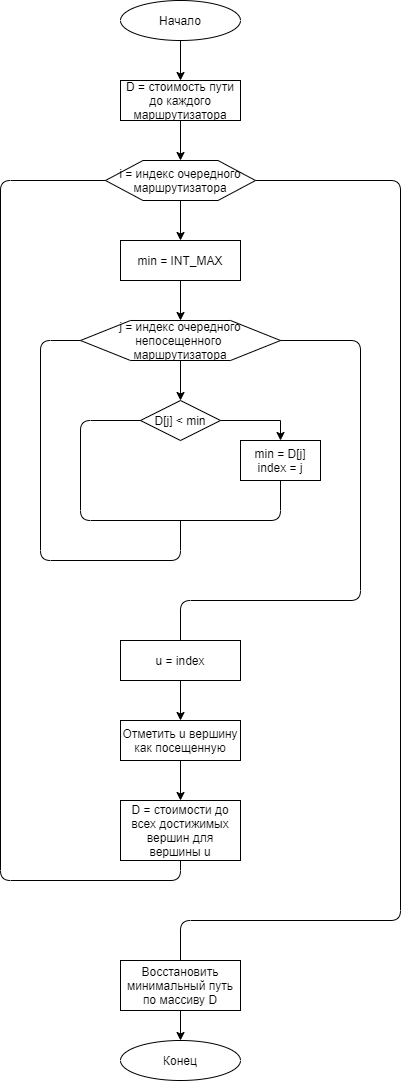
}

}

Метод последовательно перебирает все соседние роутеры и получает их идентификаторы, которые записываются в массив neigbours.

Метод void settable(QVector <QVector <int>> table) используется после обмена маршрутизаторами информации о состоянии линий связи, для формирования таблицы смежности, в которой каждый элемент этой таблицы дает информацию о стоимости пути между i-м и j-м маршрутизатором. В случае, если элемент таблицы равен нулю, то пути между такими маршрутизаторами не существует.

Метод void getOptimalRoutes() реализует алгоритм Дейкстры для получения оптимальных путей (с минимальной стоимостью) между данным маршрутизатором и всеми остальными, которые можно достигнуть. Блок-схема алгоритма Дейкстры:



Реализация алгоритма Дейкстры:

void Router::**getOptimalRoutes**(){

int st = *this*->id;

QVector <int> D; D.resize(table[0].size());

QVector <bool> visited; visited.resize(table[0].size());

*for*(int i = 0; i < table[0].size(); i++){

D[i] = table[st][i];

visited[i] = *false*;

}

D[st] = 0;

int index = 0, u = 0;

*for* (int i = 0; i < table[0].size(); i++){

int min = INT\_MAX;

*for* (int j = 0; j < table[0].size(); j++){

*if* (!visited[j] && D[j] < min){

min = D[j];

index = j;

}

}

u = index;

visited[u] = *true*;

*for*(int j = 0; j < table[0].size(); j++){

*if* (!visited[j] && table[u][j] != INT\_MAX && D[u] != INT\_MAX && (D[u] + table[u][j] < D[j])){

D[j] = D[u] + table[u][j];

}

}

}

QString vis = "";

*for* (int pos = 0; pos < table[0].size(); pos++){

qDebug() << *this*->id;

*if* (D[pos] != 0 && D[pos] != INT\_MAX || pos != *this*->id){

int end = pos;

vis += QString::number(pos);

int k = 1;

int weight = D[end];

*while* (end != st){

*for* (int i = 0; i<table[0].size(); i++)

*if* (table[i][end] != 0){

int temp = weight - table[i][end];

*if* (temp == D[i]){

weight = temp;

end = i;

vis = vis + " " + QString::number(i);

k++;

}

}

}

*//qDebug()* *<<* *vis;*

std::reverse(vis.begin(), vis.end());

optimalRoutes[D[pos]] = vis;

vis.clear();

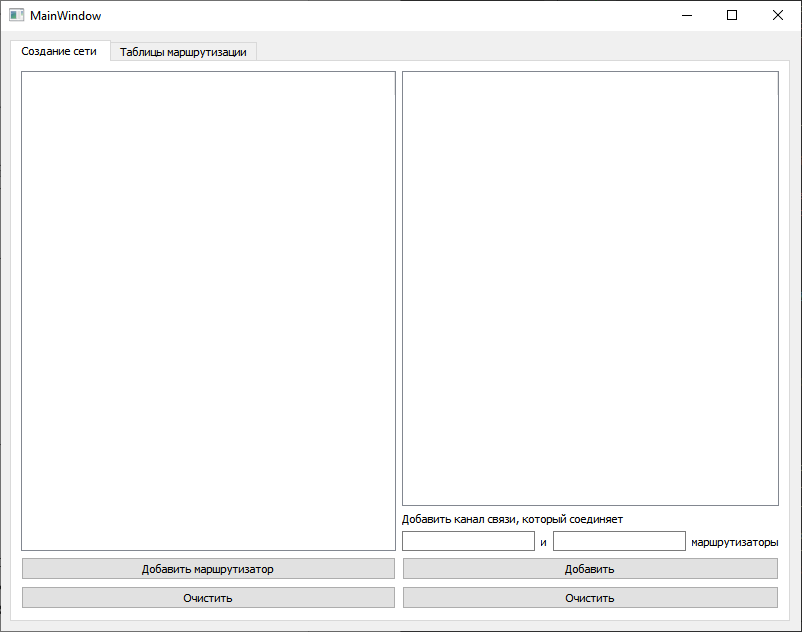
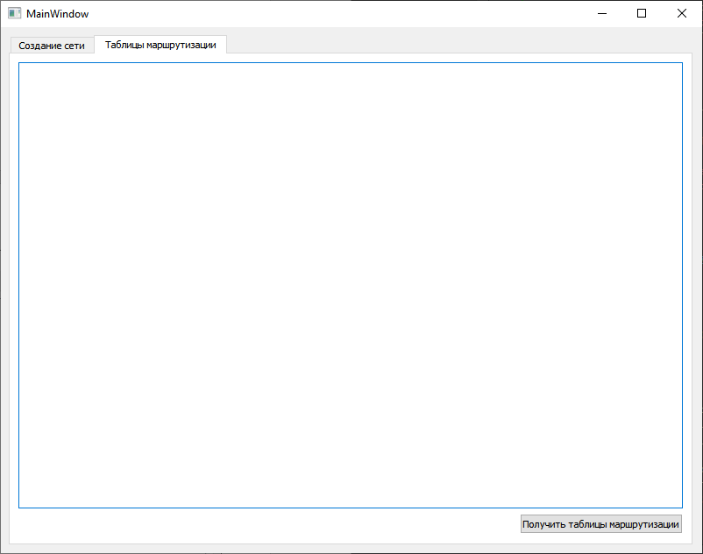
}

}

qDebug() << optimalRoutes;

}

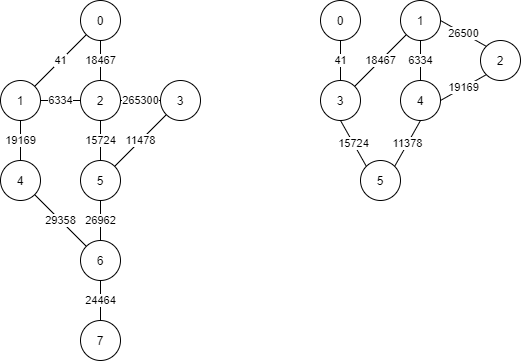
GUI разрабатываемой программы представлен в виде двух списков, в которые можно добавлять / удалять маршрутизаторы и линии связи между ними соответственно, и таблицы маршрутизации для каждого маршрутизатора:

 .

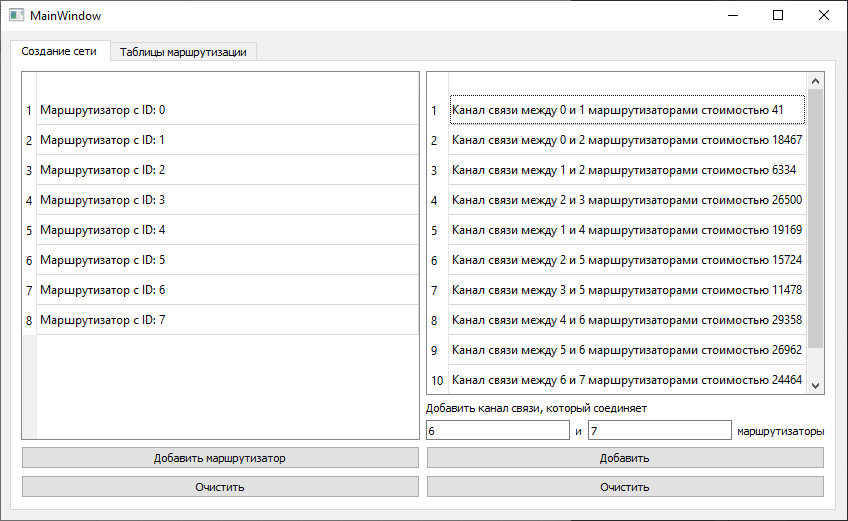
С учетом того, что данные могут быть введены некорректно, возможность «кастомизировать» вводимый граф минимальна, id роутеров и линий задаются автоматически, как и метрика линий связи.

1. **Тестирование**

Для удобства представим входные данные в виде графа. Подберем такие тестовые данные, в которых к каждому маршрутизатору будет подключено несколько линий связи.



Введем тестовые данные:



Полученный результат:

Таблица маршрутизации для 0 роутера

Маршрут с 1 маршрутизатором ценой 41 единиц: 0 1

Маршрут с 2 маршрутизатором ценой 6375 единиц: 0 1 2

Маршрут с 4 маршрутизатором ценой 19210 единиц: 0 1 4

Маршрут с 5 маршрутизатором ценой 22099 единиц: 0 1 2 5

Маршрут с 3 маршрутизатором ценой 32875 единиц: 0 1 2 3

Маршрут с 6 маршрутизатором ценой 48568 единиц: 0 1 4 6

Маршрут с 7 маршрутизатором ценой 73032 единиц: 0 1 4 6 7

Таблица маршрутизации для 1 роутера

Маршрут с 0 маршрутизатором ценой 41 единиц: 1 0

Маршрут с 2 маршрутизатором ценой 6334 единиц: 1 2

Маршрут с 4 маршрутизатором ценой 19169 единиц: 1 4

Маршрут с 5 маршрутизатором ценой 22058 единиц: 1 2 5

Маршрут с 3 маршрутизатором ценой 32834 единиц: 1 2 3

Маршрут с 6 маршрутизатором ценой 48527 единиц: 1 4 6

Маршрут с 7 маршрутизатором ценой 72991 единиц: 1 4 6 7

Таблица маршрутизации для 2 роутера

Маршрут с 1 маршрутизатором ценой 6334 единиц: 2 1

Маршрут с 0 маршрутизатором ценой 6375 единиц: 2 1 0

Маршрут с 5 маршрутизатором ценой 15724 единиц: 2 5

Маршрут с 4 маршрутизатором ценой 25503 единиц: 2 1 4

Маршрут с 3 маршрутизатором ценой 26500 единиц: 2 3

Маршрут с 6 маршрутизатором ценой 42686 единиц: 2 5 6

Маршрут с 7 маршрутизатором ценой 67150 единиц: 2 5 6 7

Таблица маршрутизации для 3 роутера

Маршрут с 5 маршрутизатором ценой 11478 единиц: 3 5

Маршрут с 2 маршрутизатором ценой 26500 единиц: 3 2

Маршрут с 1 маршрутизатором ценой 32834 единиц: 3 2 1

Маршрут с 0 маршрутизатором ценой 32875 единиц: 3 2 1 0

Маршрут с 6 маршрутизатором ценой 38440 единиц: 3 5 6

Маршрут с 4 маршрутизатором ценой 52003 единиц: 3 2 1 4

Маршрут с 7 маршрутизатором ценой 62904 единиц: 3 5 6 7

Таблица маршрутизации для 4 роутера

Маршрут с 1 маршрутизатором ценой 19169 единиц: 4 1

Маршрут с 0 маршрутизатором ценой 19210 единиц: 4 1 0

Маршрут с 2 маршрутизатором ценой 25503 единиц: 4 1 2

Маршрут с 6 маршрутизатором ценой 29358 единиц: 4 6

Маршрут с 5 маршрутизатором ценой 41227 единиц: 4 1 2 5

Маршрут с 3 маршрутизатором ценой 52003 единиц: 4 1 2 3

Маршрут с 7 маршрутизатором ценой 53822 единиц: 4 6 7

Таблица маршрутизации для 5 роутера

Маршрут с 3 маршрутизатором ценой 11478 единиц: 5 3

Маршрут с 2 маршрутизатором ценой 15724 единиц: 5 2

Маршрут с 1 маршрутизатором ценой 22058 единиц: 5 2 1

Маршрут с 0 маршрутизатором ценой 22099 единиц: 5 2 1 0

Маршрут с 6 маршрутизатором ценой 26962 единиц: 5 6

Маршрут с 4 маршрутизатором ценой 41227 единиц: 5 2 1 4

Маршрут с 7 маршрутизатором ценой 51426 единиц: 5 6 7

Таблица маршрутизации для 6 роутера

Маршрут с 7 маршрутизатором ценой 24464 единиц: 6 7

Маршрут с 5 маршрутизатором ценой 26962 единиц: 6 5

Маршрут с 4 маршрутизатором ценой 29358 единиц: 6 4

Маршрут с 3 маршрутизатором ценой 38440 единиц: 6 5 3

Маршрут с 2 маршрутизатором ценой 42686 единиц: 6 5 2

Маршрут с 1 маршрутизатором ценой 48527 единиц: 6 4 1

Маршрут с 0 маршрутизатором ценой 48568 единиц: 6 4 1 0

Таблица маршрутизации для 7 роутера

Маршрут с 6 маршрутизатором ценой 24464 единиц: 7 6

Маршрут с 5 маршрутизатором ценой 51426 единиц: 7 6 5

Маршрут с 4 маршрутизатором ценой 53822 единиц: 7 6 4

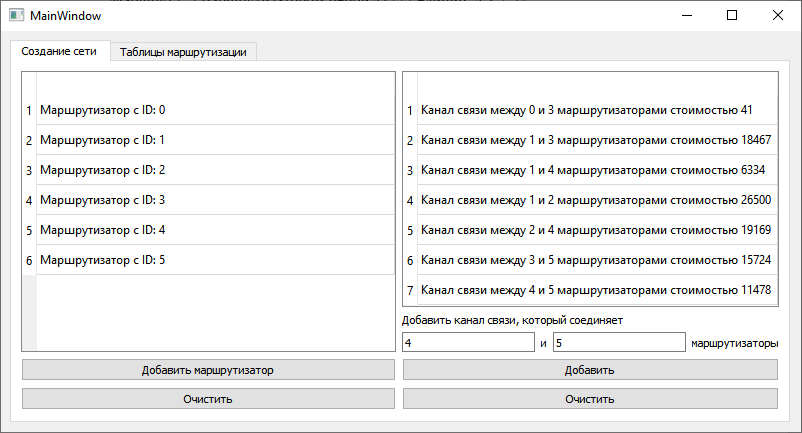
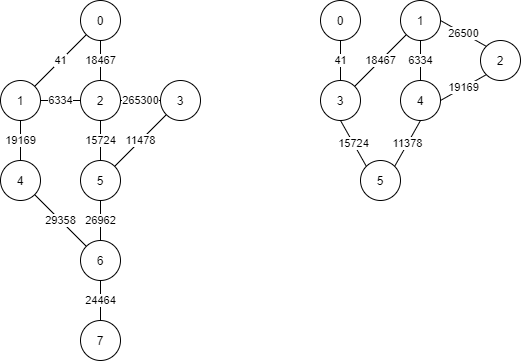
Маршрут с 3 маршрутизатором ценой 62904 единиц: 7 6 5 3

Маршрут с 2 маршрутизатором ценой 67150 единиц: 7 6 5 2

Маршрут с 1 маршрутизатором ценой 72991 единиц: 7 6 4 1

Маршрут с 0 маршрутизатором ценой 73032 единиц: 7 6 4 1 0

Попробуем другие данные:



В итоге получим:

Таблица маршрутизации для 0 роутера

Маршрут с 3 маршрутизатором ценой 41 единиц: 0 3

Маршрут с 5 маршрутизатором ценой 15765 единиц: 0 3 5

Маршрут с 1 маршрутизатором ценой 18508 единиц: 0 3 1

Маршрут с 4 маршрутизатором ценой 24842 единиц: 0 3 1 4

Маршрут с 2 маршрутизатором ценой 44011 единиц: 0 3 1 4 2

Таблица маршрутизации для 1 роутера

Маршрут с 4 маршрутизатором ценой 6334 единиц: 1 4

Маршрут с 5 маршрутизатором ценой 17812 единиц: 1 4 5

Маршрут с 3 маршрутизатором ценой 18467 единиц: 1 3

Маршрут с 0 маршрутизатором ценой 18508 единиц: 1 3 0

Маршрут с 2 маршрутизатором ценой 25503 единиц: 1 4 2

Таблица маршрутизации для 2 роутера

Маршрут с 4 маршрутизатором ценой 19169 единиц: 2 4

Маршрут с 1 маршрутизатором ценой 25503 единиц: 2 4 1

Маршрут с 5 маршрутизатором ценой 30647 единиц: 2 4 5

Маршрут с 3 маршрутизатором ценой 43970 единиц: 2 4 1 3

Маршрут с 0 маршрутизатором ценой 44011 единиц: 2 4 1 3 0

Таблица маршрутизации для 3 роутера

Маршрут с 0 маршрутизатором ценой 41 единиц: 3 0

Маршрут с 5 маршрутизатором ценой 15724 единиц: 3 5

Маршрут с 1 маршрутизатором ценой 18467 единиц: 3 1

Маршрут с 4 маршрутизатором ценой 24801 единиц: 3 1 4

Маршрут с 2 маршрутизатором ценой 43970 единиц: 3 1 4 2

Таблица маршрутизации для 4 роутера

Маршрут с 1 маршрутизатором ценой 6334 единиц: 4 1

Маршрут с 5 маршрутизатором ценой 11478 единиц: 4 5

Маршрут с 2 маршрутизатором ценой 19169 единиц: 4 2

Маршрут с 3 маршрутизатором ценой 24801 единиц: 4 1 3

Маршрут с 0 маршрутизатором ценой 24842 единиц: 4 1 3 0

Таблица маршрутизации для 5 роутера

Маршрут с 4 маршрутизатором ценой 11478 единиц: 5 4

Маршрут с 3 маршрутизатором ценой 15724 единиц: 5 3

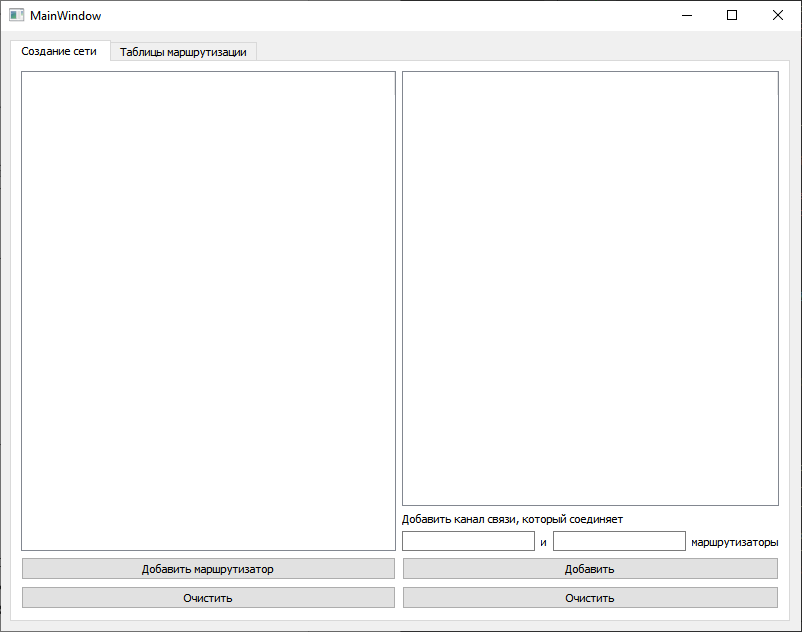
Маршрут с 0 маршрутизатором ценой 15765 единиц: 5 3 0

Маршрут с 1 маршрутизатором ценой 17812 единиц: 5 4 1

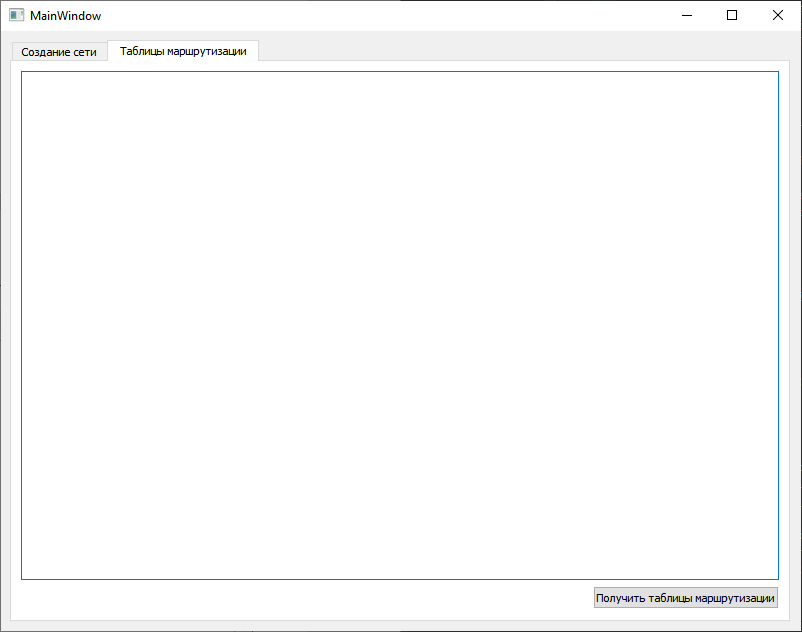
Маршрут с 2 маршрутизатором ценой 30647 единиц: 5 4 2

# **Руководство пользователя**

Для ввода данных о сети маршрутизаторов пользователь должен открыть вкладку «Создание сети», и добавить маршрутизаторы и связи между ними. В случае, если были введены данные с ошибкой, их можно удалить, выделив их в списке и нажать кнопку «Очистить».



Для получения таблицы маршрутизации достаточно перейти на вкладку «Таблицы маршрутизации» и нажать кнопку «Получить таблицы маршрутизации».



# **Заключение**

В процессе выполнения курсовой работы были изучены алгоритмы маршрутизации в сети и реализована программа, моделирующая алгоритм маршрутизации с учетом состояния линий связи на языке с++ в среде разработки Qt. В процессе изучения данного вопроса был проведен сравнительный анализ алгоритмов маршрутизации в сети, выявлены достоинства и недостатки некоторых из них.

Приведя классификацию алгоритмов маршрутизации и рассмотрев некоторые из них, можно прийти к выводу, что каждый из них обладает своими достоинствами и недостатками, рассмотренными выше, но при этом применяются в различных протоколах маршрутизации (RIP, BGP, IS-IS, OSPF, NLSP и пр.).

Реализованный алгоритм является адаптивным, что характеризует его как алгоритм, который гарантирует равномерное распределение нагрузки по сети, т. к. используются пути с минимальной стоимостью, а также достаточно гибким относительно изменения топологии сети. Одним из недостатков алгоритма с учетом состояния линий связи можно назвать необходимость в наличии некоторых вычислительных мощностей у каждого маршрутизатора, чтобы обеспечить вычисление кратчайших путей, составление таблицы маршрутизации, и некоторый объем дополнительной памяти, в котором таблица маршрутизации будет хранится.

# **Список использованных источников**

1. Алгоритмы маршрутизации – Википедия – [Электронный ресурс] – URL: https://ru.wikipedia.org/wiki/Алгоритмы\_маршрутизации
2. Протоколы маршрутизации – Википедия – [Электронный ресурс] – URL: https://ru.wikipedia.org/wiki/Протоколы\_маршрутизации
3. Routing algorithms – Javapoint – [Электронный ресурс] – URL: https://www.javatpoint.com/computer-network-routing-algorithm
4. Qt Documentation – [Электронный ресурс] – URL: https://doc.qt.io/

# **Приложения**

**Содержимое файла mainwindow.h:**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QVector>

#include <QMap>

#include <QTableWidget>

#include <QDebug>

#include <QMessageBox>

#include <QMultiMap>

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class Route{

private:

int id;

int cost;

bool isAvaliable;

QVector <int> connectedRouters;

public:

Route(int id):id(id){cost = rand();};

int getID(){return this->id;}

int getCost(){return this->cost;}

bool connectToRouter(int routerID);

int getNeighbour(int);

QVector <int> getConnectedRouters(){return this->connectedRouters;}

};

class Router{

private:

int id;

QVector <Route \*> routes;

QVector <int> neighbours;

QVector <QVector <int>> table;

public:

QMap<int, QString> optimalRoutes;

Router(int id):id(id){};

int getID(){return id;}

void sendHello();

bool connectToRoute(Route \* r);

QVector <int> getNeighbours(){return this->neighbours;};

int getCostTo(int routerID);

void getOptimalRoutes();

void setTable(QVector <QVector <int>> table);

QString getRoutingTable();

};

class Network{

private:

QVector <Route \*> routes;

QVector <Router \*> routers;

QVector <QVector <int>> table;

public:

void setRouters(QVector <Router \*> routers);

void setRoutes(QVector <Route \*> routes);

void getTable();

void sendTable();

};

class MainWindow : public QMainWindow{

Q\_OBJECT

private:

Network \* n;

int routersCount = 0;

int routesCount = 0;

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private slots:

void on\_addRouter\_clicked();

void on\_addRoute\_clicked();

void on\_pushButton\_clicked();

private:

QVector<Route \*> routes;

QVector<Router \*> routers;

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

**Содержимое файла mainwindow.cpp:**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QDebug>

MainWindow::MainWindow(QWidget \*parent) : QMainWindow(parent), ui(new Ui::MainWindow){

ui->setupUi(this);

n = new Network();

ui->routersTable->setColumnCount(1);

ui->routersTable->setRowCount(0);

ui->routersTable->setHorizontalHeaderItem(0, new QTableWidgetItem(""));

ui->routesTable->setColumnCount(1);

ui->routesTable->setHorizontalHeaderItem(0, new QTableWidgetItem(""));

ui->routersTable->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);

ui->routesTable->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);

//ui->routingTable->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);

}

MainWindow::~MainWindow(){

delete n;

delete ui;

}

bool Route::connectToRouter(int routerID){

if (connectedRouters.size() > 2) return false;

connectedRouters.append(routerID);

return true;

}

int Route::getNeighbour(int currentRouter){

for (int i = 0; i < connectedRouters.size(); i++){

if (currentRouter != connectedRouters[i]) return connectedRouters[i];

}

return -1;

}

bool Router::connectToRoute(Route \* r){

if (routes.contains(r)) return false;

routes.append(r);

r->connectToRouter(this->id);

return true;

}

void Router::sendHello(){

for (int i = 0; i < routes.size(); i++){

neighbours.append(routes[i]->getNeighbour(this->id));

}

}

int Router::getCostTo(int routerID){

for (int i = 0; i < routes.size(); i++){

QVector <int> tmp = routes[i]->getConnectedRouters();

if (tmp.contains(routerID) && tmp.contains(this->id))

return routes[i]->getCost();

}

return -1;

}

void Router::setTable(QVector<QVector<int> > table){

this->table = table;

}

void Router::getOptimalRoutes(){

int st = this->id;

QVector <int> D; D.resize(table[0].size());

QVector <bool> visited; visited.resize(table[0].size());

for(int i = 0; i < table[0].size(); i++){

D[i] = table[st][i];

visited[i] = false;

}

D[st] = 0;

int index = 0, u = 0;

for (int i = 0; i < table[0].size(); i++){

int min = INT\_MAX;

for (int j = 0; j < table[0].size(); j++){

if (!visited[j] && D[j] < min){

min = D[j];

index = j;

}

}

u = index;

visited[u] = true;

for(int j = 0; j < table[0].size(); j++){

if (!visited[j] && table[u][j] != INT\_MAX && D[u] != INT\_MAX && (D[u] + table[u][j] < D[j])){

D[j] = D[u] + table[u][j];

}

}

}

QString vis = "";

for (int pos = 0; pos < table[0].size(); pos++){

qDebug() << this->id;

if (D[pos] != 0 && D[pos] != INT\_MAX || pos != this->id){

int end = pos;

vis += QString::number(pos);

int k = 1;

int weight = D[end];

while (end != st){

for (int i = 0; i<table[0].size(); i++)

if (table[i][end] != 0){

int temp = weight - table[i][end];

if (temp == D[i]){

weight = temp;

end = i;

vis = vis + " " + QString::number(i);

k++;

}

}

}

//qDebug() << vis;

std::reverse(vis.begin(), vis.end());

optimalRoutes[D[pos]] = vis;

vis.clear();

}

}

qDebug() << optimalRoutes;

}

QString Router::getRoutingTable(){

qDebug() << "!";

QString result = "";

QMutableMapIterator<int, QString> i(this->optimalRoutes);

while (i.hasNext()){

i.next();

int cost = i.key();

QString route = i.value();

QStringList tmp = route.split(" ");

QString with = tmp[tmp.size() - 1];

result += QString("Маршрут с " + with + " маршрутизатором ценой %1 единиц: " + route + "\n").arg(cost);

}

qDebug() << result;

return result;

}

void Network::setRoutes(QVector<Route \*> routes){

this->routes = routes;

}

void Network::setRouters(QVector<Router \*> routers){

this->routers = routers;

}

void Network::getTable(){

table.resize(routers.size());

for (int i = 0; i < routers.size(); i++){

table[i].resize(routers.size());

table[i].fill(INT\_MAX);

QVector <int> tmp = routers[i]->getNeighbours();

for (int j = 0; j < tmp.size(); j++){

table[i][tmp[j]] = routers[i]->getCostTo(tmp[j]);

}

}

//qDebug() << table;

}

void Network::sendTable(){

for (int i = 0; i < routers.size(); i++){

routers[i]->setTable(this->table);

}

}

void MainWindow::on\_addRouter\_clicked(){

routers.append(new Router(routersCount));

ui->routersTable->setRowCount(routersCount + 1);

ui->routersTable->setItem(routersCount, 0,

new QTableWidgetItem("Маршрутизатор c ID: " + QString::number(routersCount)));

routersCount += 1;

}

void MainWindow::on\_addRoute\_clicked(){

if (ui->router1id->text() == "" || ui->router2id->text() == ""){

QMessageBox::warning(this, "Ну вот опять!", "Вы не ввели ID маршрутизаторов!");

return;

}

routes.append(new Route(routesCount));

int a = ui->router1id->text().toInt();

int b = ui->router2id->text().toInt();

routes[routesCount]->connectToRouter(a);

routes[routesCount]->connectToRouter(b);

ui->routesTable->setRowCount(routesCount + 1);

ui->routesTable->setItem(routesCount, 0,

new QTableWidgetItem(

QString("Канал связи между %1 и %2 маршрутизаторами").arg(a).arg(b)));

routers[a]->connectToRoute(routes[routesCount]);

routers[b]->connectToRoute(routes[routesCount]);

routesCount++;

}

void MainWindow::on\_pushButton\_clicked(){

for (int i = 0; i < routers.size(); i++){

routers[i]->sendHello();

}

n->setRoutes(routes);

n->setRouters(routers);

n->getTable();

n->sendTable();

//ui->routingTable->setRowCount(routersCount);

for (int i = 0; i < routers.size(); i++){

routers[i]->getOptimalRoutes();

QString newRouterString = QString("Таблица маршрутизации для %1 роутера\n").arg(i);

newRouterString += routers[i]->getRoutingTable();

ui->routingTable->append(newRouterString);

qDebug() << newRouterString;

}

}