

ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ И НАУКИ ГОРОДА МОСКВЫ

Государственное бюджетное общеобразовательное учреждение  
 города Москвы

**«Школа № 1533 «ЛИТ»**

**ВЫПУСКНОЙ ПРОЕКТ**

**Профиль**: Программирование

**Тема**: Исследование кольцевых туристических маршрутов на сети железных дорог в России

**Исполнитель:**

Сергунов Анатолий Дмитриевич, группа 10.3

**Заказчик:**

Департамент информатизации ОАО “РЖД”

**Руководитель:**

Гиглавый Александр Владимирович, кандидат наук

Москва, 2023 г

Оглавление

[Введение 4](#_Toc137738768)

[Постановка задачи 4](#_Toc137738769)

[Актуальность и целевая аудитория 5](#_Toc137738770)

[Актуальность 5](#_Toc137738771)

[Целевая аудитория 5](#_Toc137738772)

[Анализ предметной области 7](#_Toc137738773)

[Гео-составляющая проекта 7](#_Toc137738774)

[Использование теории графов 7](#_Toc137738775)

[Обзор аналогов 9](#_Toc137738776)

[German Fairy Tail Rail 9](#_Toc137738777)

[В Карелию 10](#_Toc137738778)

[В Великий Устюг 10](#_Toc137738779)

[Жемчужина Кавказа 11](#_Toc137738780)

[Переславский экспресс 11](#_Toc137738781)

[Сравнительная таблица 12](#_Toc137738782)

[Проект Александра Фролова 12](#_Toc137738783)

[Решение 14](#_Toc137738784)

[Теоретические выкладки 14](#_Toc137738785)

[Алгоритмы 14](#_Toc137738786)

[Программная реализация 14](#_Toc137738787)

[Ход работы 16](#_Toc137738788)

[Анализ задачи Александром Фроловым 16](#_Toc137738789)

[Соотнесение станций и исторических поселений 16](#_Toc137738790)

[Анализ перспектив алгоритмического построения колец 17](#_Toc137738791)

[Процесс формулировку постановки задачи 18](#_Toc137738792)

[Первая попытка алгоритмического получения данных 18](#_Toc137738793)

[Вторая попытка алгоритмического получения данных 20](#_Toc137738794)

[Попытка построения математической модели для получения данных 20](#_Toc137738795)

[Расширение информации о железнодорожных станциях 21](#_Toc137738796)

[Получение данных с помощью геокодирования 22](#_Toc137738797)

[Создание базы данных 22](#_Toc137738798)

[Попытка создания математической модели построения кольцевых маршрутов 24](#_Toc137738799)

[Алгоритм построения кольцевых маршрутов 26](#_Toc137738800)

[Визуализация 28](#_Toc137738801)

[Результат 29](#_Toc137738802)

[Решение поставленной задачи 29](#_Toc137738803)

[Исследование полученных колец 29](#_Toc137738804)

# Введение

За последние несколько лет в России возрос интерес к внутреннему туризму. Вследствие этого возникла необходимость в развитии туристических маршрутов для российского транспорта, в частности – для железных дорог. Сперва для разработки туристических маршрутов для РЖД были задействованы турагентства – те разработали различные туристические маршруты (минусы многих заключались в долгом движении в одну сторону ради одного интересного для туристов города). Параллельно в различных мировых турагентствах появляются идеи об удобстве такого типа туристических маршрутов как кольцевые (начинающиеся и заканчивающиеся в одном городе). После анализа железных дорог Европейской части России выяснилось, что плотность исторических городов (представляющих интерес для туристов) на ней очень высока, следовательно, есть возможность создать множество приемлемых туристических кольцевых маршрутов. Так появилась потребность в создании алгоритма, занимающегося построением туристических кольцевых маршрутов на Европейской части России.

# Постановка задачи

Входные данные проекта – таблица – список всех станций Российских железных дорог (с указаниями широты и долготы), список исторических поселений (городов, представляющих интерес для туристов), а также нарисованная в ГИС-приложении QGIS карта железных дорог. Сама задача проекта условно делится на 3 части: обработка данных, алгоритм построения маршрутов и визуализация. Задача обработки данных заключается в том, чтобы соотнести исторические поселения со станциями, извлечь из карты в QGIS’е данные о графе железных дорог и представить их в виде списка векторов (где вектор – отрезок от одной узловой станции до другой). Алгоритм построения маршрутов обрабатывает полученные данные, создает вокруг каждой подходящей по критериям интереса для туристов столицы субъекта РФ кольцевой маршрут и представляет его в виде последовательного списка векторов. Задача визуализации заключается в изображении этих маршрутов в ГИС-приложении QGIS.

# Актуальность и целевая аудитория

## Актуальность

В ходе событий последних лет, несмотря на серьезный урон внешнему туризму со стороны глобальных мировых событий, внутреннему российскому туризму прогнозируют рост популярности (по последним аналитикам, анализирующим состояние туристического рынка в конец пандемии, популярность внутреннего туризма значительно возросла по сравнению с 2019 годом).

Вместе с тем идея именно кольцевого маршрута распространена по всему миру: в частности в Германии турагентство разработало маршрут, основанный на сказках братьев Гримм. Между тем идея туристических кольцевых маршрутов применима именно к Российским железным дорогам. Это связано с несколькими факторами. Во-первых, Россия имеет одну из наиболее разветвленных железнодорожных сетей в мире, что позволяет создавать огромное число вариаций железнодорожных маршрутов (от однодневных до многодневных). Во-вторых, Россия наполнена интересными для туризма городами: в перечне исторических городов России от Министерства Культуры РФ за 2002 год (список, на который во многом ориентируется проект) насчитывается 478 населенных пунктов, при этом более 150 таковых поселений лежат на железнодорожной сети Европейской части России. Это позволяет формировать здесь интересные с точки зрения туризма маршруты. В-третьих, население России по меркам Европы достаточно высоко, в большинстве столицах субъектов РФ проживает достаточно большое число человек, чтобы для них был актуален запуск туристического кольцевого маршрута.

## Целевая аудитория

Основная аудитория, на которую рассчитаны полученные маршруты, - учащиеся возраста от 12 до 18 лет. Разумеется, подразумевается, что маршруты будут актуальные и для учащихся старших возрастов, вплоть до 25 лет, а также не только для учащихся. Однако 12-18 лет это тот возраст, когда человек уже способен на искренний интерес к чему-то, связанному с историей и культурой, и вместе с тем это возраст, когда человек еще не сильно загружен учебой, следовательно, имеет время на путешествия по родной стране.

При этом это весьма новая аудитория для мирового туризма. На основе типологии туристов, созданной немецким исследователем Ганом, школьников можно отнести к так называемому B-типу (образование и обзор достопримечательностей). Данный тип Ган подразделяет на 3 подтипа: “эксперты”, “коллекционирующие” посещаемые достопримечательности; “эмоциональные” любители культуры и природы; “специалисты”, углубляющие свои знания в определенных областях культуры. Соотнося данные подтипы со школьной аудиторией, можем предположить, что отдельных школьников в зависимости от их интересов и познаний можно отнести к одному из трех вышеуказанных подтипов. Уточняя, более сомнительным представляется отнесение школьников к первой категории, поскольку перспектива “коллекционирования достопримечательностей” более характерна для туристов старшего возраста. Однако вторая и третья категории являются весьма актуальными в разговоре о школьной аудитории. Иногда интерес к туристическим маршрутам основывается у школьников на получении ярких эмоций, связанных с достопримечательностями или красотами Родины. Иногда же, когда школьник живо заинтересован в какой-то конкретной области истории или культуры, он надеется в каком-то конкретном туристическом маршруте пополнить свой запас знаний в этой области.

Таким образом, несмотря на то, что школьная аудитория не является достаточно живо освоенной мировым туризмом, в типологии туристов, основанной на мировых исследованиях, есть место для школьников как относящихся к определенному типу туристов.

# Анализ предметной области

## Гео-составляющая проекта

Кольцо (кольцевой маршрут) – маршрут, начинающийся и заканчивающийся в одной точке.

Зона (субъекта РФ) – окружающее субъект РФ пространство, состоящее из прилегающих к нему субъектов. Поиск кольцевого маршрута проходит в каждой из зон для подходящих субъектов.

Узловая станция – станция, соединенная напрямую более чем с двумя другими станциями.

Смежные узловые станции – две узловые станции, соединённые таким образом, что между ними нет третьей узловой станции.

Вектор – две соединенные смежные узловые станции.

Критерии адекватности кольцевого маршрута (исходя из целевой аудитории и плотности исторических городов в ЕЧ): кольцевой маршрут занимает 4-5 дней пути, маршрут проходит через интересные туристам города (исторические поселения или другие примечательные места России), по большей части принадлежащие тому субъекту, в котором происходит поиск колец, максимальная разница в широте маршрута и максимальная разница в долготе маршрута не должны сильно различаться (кодом реализовано, что частное деления большего из этих чисел на меньшее не должно превосходить 3).

В проекте идет работа со следующими географическими объектами: станциями (имеющими широту и долготу). Эта работа требует базовых знаний геокодирования. Геокодирование – процесс назначения географических идентификаторов (в нашем случае широты и долготы) объектам карты и записям данных.

ГИС-приложение – приложение, основной функционал которого занимает геокодирование, служит для наглядного представления географических данных. В проекте также будет служить для редактирования конечных данных – кольцевых маршрутов.

## Использование теории графов

Работа в частности основана на теории графов и на представлении графа в векторном пространстве. Железнодорожная сеть является графом, но при этом для задачи необходимо учитывать координаты железнодорожных станций (вершин графа) и направление вектора (не направление с точки зрения теории графов, а угол вектора с горизонталью или вертикалью). Таким образом, данные представлены в виде списка векторов – соединенных узловых вершин графа, то есть простым запросом к базе данных для узловой вершины можно получить список смежности (в теории графов – список для каждой точки графа смежных с ней вершин). Работа с векторами во многом основана на углах между вектором и вертикалью, а также углах между двумя векторами: таким образом, алгоритм выбирает наиболее близкий вектор к идеальному.

# Обзор аналогов

## German Fairy Tail Rail

Прямых аналогов проекта, реализованных кодом нет. Все туристические кольцевые маршруты, разработанные на данный момент, являются результатом ручной работы турагентств, основанной в первую очередь не на особенностях путешествия по железным дорогам, а на какой-то тематической составляющей маршрута.

Самой известной зарубежной работой турагентств, послужившей одним из доказательств актуальности кольцевых туристических маршрутов для России, является маршрут “8-Day German Fairy Tale Rail Circle Tour”, представляющий из себя 8-дневную поездку по городам, упоминаемым в сказках братьев Гримм.

Турагентства, сотрудничающие с РЖД, разработали много туристических маршрутов – некоторые из них даже имеют в основе кольцевую модель. Ниже представлены некоторые из них.

## В Карелию

Первый – маршрут под названием “В Карелию”. Он основан на довольно сомнительной, но вместе с тем кольцевой модели, и выглядит следующим образом: Москва – Петрозаводск – Сортавала – Выборг – Москва. Маршрут этот на сайте РЖД назван “туристической классикой”. Между тем совершенно очевидно, что несмотря на понятную концепцию маршрута, он имеет весьма значительный повод быть подверженным критике. Концепция же маршрута основана на знакомстве москвичей с таким весьма знаменательным краем России как Карелия. Основная идея маршрута – любоваться этим “краем лесов и озер”. Маршрут затрагивает 5 исторические местности: помимо трех вышеуказанных старинных городов, предлагается включенное в маршрут путешествие до резиденции Деда Мороза – зимой – или до музея-заповедника “Кижи” – летом, а также до горного парка “Рускеала” – достопримечательности Карелии, известной за “живописные мраморные скалы”. Между тем, исходя из вышеописанного анализа железных дорог Европейской части России, маршрут кажется сомнительным. Во-первых, большую часть маршрута занимает длинный, неинтересный с точки зрения туристов путь от Москвы до Карелии и обратно. Если вспоминать о более 150 исторических поселениях на Европейской части России, имеющих доступ к железной дороге, то кажется нерациональным использованием этого богатства России интересными городами. Однако же, если принять во внимание, что маршрут занимает 2 дня и затрагивает 5 интересных для туристов точек, то, разумеется, плохим его назвать трудно. Но все же, пользуясь кольцевой моделью, можно создать туристические маршруты, которые охватывают значительно большее количество исторических городов, что и подтверждается результатами этого проекта.

## В Великий Устюг

Еще более сомнителен маршрут “Русский Север. В Великий Устюг”. Весь маршрут заключается в трехдневном путешествии из Москвы до Великого Устюга и обратно. Разумеется, хотя формально маршрут соответствует кольцевой модели, очевидно, что он очень нерационально использует богатство исторических городов на территории России. Также сложно говорить и о какой-либо тематической концепции маршрута, хотя формально как раз маршрут весьма легко в концепцию укладывается: “Русский Север” – так объясняется ценность этого маршрута. Между тем данный маршрут очень сложно анализировать с точки зрения полноценного туристического маршрута, поскольку основой любого туристического маршрута является насыщенность историческими точками маршрута. Здесь же таковая только одна.

## Жемчужина Кавказа

Следующий маршрут – “Жемчужина Кавказа” – построен более интересно с точки зрения данной задачи. Маршрут составлен так: Москва – Майкоп – Владикавказ – Грозный – Дербент – Нальчик – Москва. Несмотря на достаточно долгий неинтересный путь от Москвы до Кавказа и обратно, этот маршрут более проработан с точки зрения исторических городов. И хотя концепция маршрута описывается в первую очередь как “увидеть живописную природу Кавказа”, этот маршрут представляет интерес туриста, который желает разбираться в культуре или истории Родины, значительно больший, чем любой другой туристический маршрут по Кавказу. Это связано с тем, что туризм по Кавказу зачастую курортного типа: местные турагентства прокладывают маршруты по самым популярным среди отдыхающих городам Кавказа – то есть по городам-курортам. Этот же маршрут основан именно на городах, которые представляют исторический интерес. Единственным его существенным недостатком можно выделить продолжительность: маршрут длится 7 дней, причем почти 2 из этих 7 дней занимает путь от Москвы до Кавказа и обратно. Разумеется, гораздо рациональнее было бы организовать подобный маршрут для более приближенных к Кавказу субъектов РФ, тем более что там тоже наберется достаточное число жителей, заинтересованных в подобной поездке. В этом плане более выигрышным вариантом является целевая аудитория маршрутов данного проекта. И конечно, пользуясь кольцевой моделью, можно реализовать аналогичные маршруты без длинного, однообразного и неинтересного движения в одном направлении (в данном случае – из Москвы в Кавказ и обратно).

## Переславский экспресс

Также в процессе разбора маршрутов, основанных на городах, представляющих интерес туристов из-за их исторической ценности, был обнаружен так называемый “Переславский экспресс”. Это достаточно короткий туристический маршрут, устроенный следующим образом: Москва – Сергиев Посад – Александров – Берендеево – Переславль-Залесский. Во-первых, несмотря на явную историческую ценность всех городов этого маршрута, туристический интерес основан лишь на Переславле-Залесском, а прочие упомянутые города проезжаются проездом с пятиминутными остановками. То есть несмотря на то, что маршрут является очень удачным и перспективным с точки зрения полноценного туристического маршрута, его потенциал не использован полностью. Также очевидным отличием этого маршрута от описываемых ранее является отсутствие кольцевой модели маршрута. Если разбирать задумку РЖД, то возвращается турист в Москву точно таким же путем после различных экскурсий по Переславлю-Залесскому, то есть путем, который ему уже не представляется ярким и интересным. Однако такова особенность Переславля-Залесского как железнодорожной станции: несмотря на то, что он представляется весьма перспективным как интересный туристам город, построение кольцевого маршрута с его участием не представляется возможным.

## Сравнительная таблица

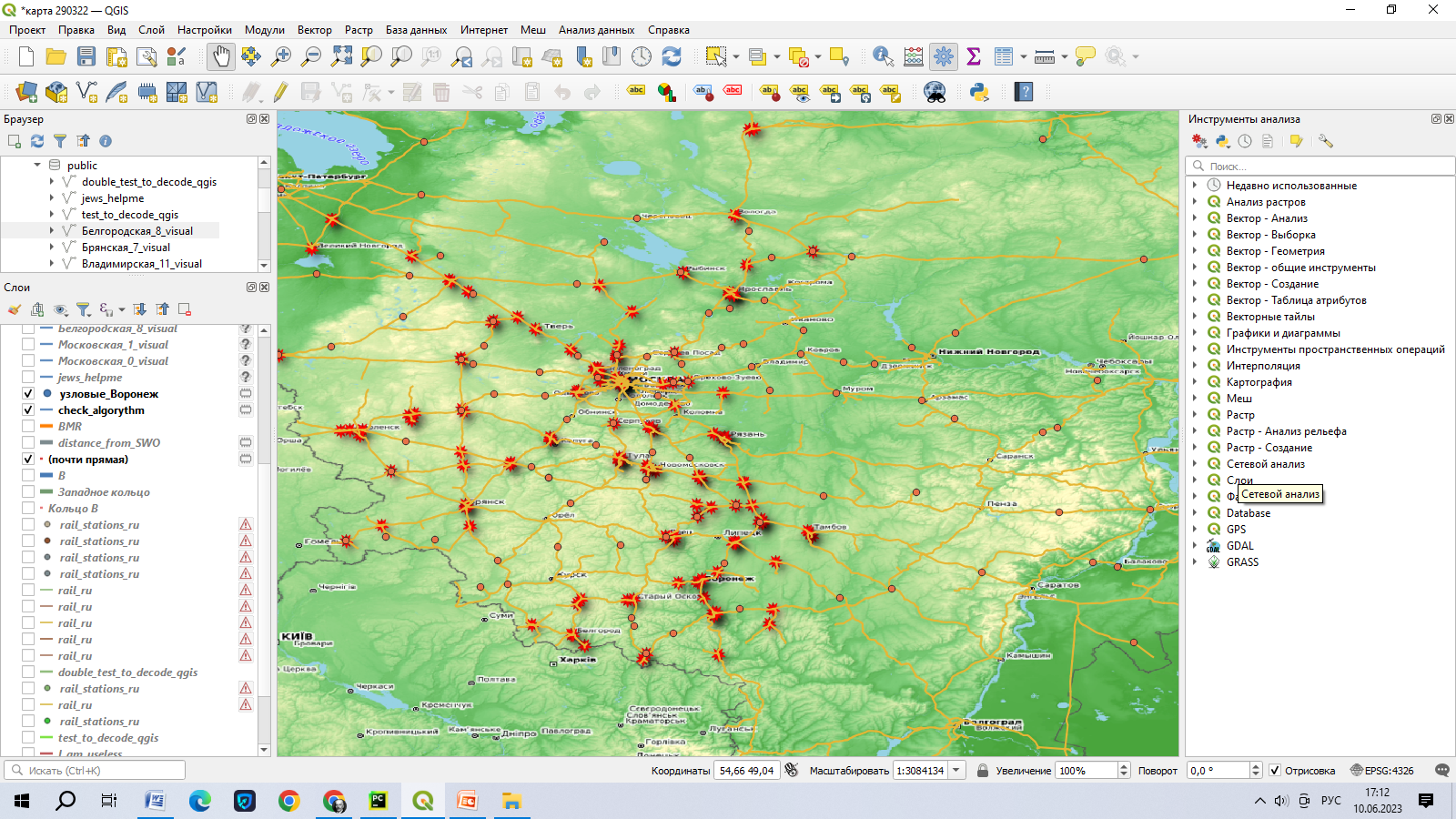
Таким образом, если проанализировать вышеперечисленные аналоги, можно получить следующую таблицу:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Насыщенность  маршрута историческими поселениями | Тематическая целостность маршрута | Кольцевая модель маршрута | Учет особенностей железных дорог | Алгоритмический поиск маршрута |
| German Fairy Tail Rail | + | + | + | + | - |
| В Карелию | - | + | + | - | - |
| В Великий Устюг | - | + | + | - | - |
| Жемчужина Кавказа | + | - | + | + | - |
| Переславский экспресс | + | + | - | + | - |
| Проект | + | - | + | +- | + |

Разумеется, проект проигрывает турагентствам по критерию “тематическая целостность маршрута”, но он не является в туристическом маршруте обязательным, если маршрут действительно представляет какой-то интерес для любителей истории и культуры. Более того, алгоритм проекта построен таким образом, что ищет интересные города в первую очередь в одном субъекте РФ и окружающей его зоне. А интересные города, расположенные в одном субъекте, вполне могут исторически быть объединены, что вполне может сложиться в настоящую тематическую целостность маршрута. Например, полученное в ходе работы алгоритма кольцо в Тульской области затрагивает как станцию Ясная Поляна (известную как одноименной усадьбой Толстого, так и исторической станцией “Козлова Засека”), так и станцию Лев Толстой – станцию, на которой умер писатель. Таким образом, данное кольцо вполне возможно тематически связать с судьбой, жизнью, памятными местами Льва Николаевича Толстого.

## Проект Александра Фролова

Если же говорить о попытках решения проблемы составления кольцевых маршрутов кодом, то этот проект является прямым продолжением работы ученика лицея Александра Фролова, который занимался этой темой в 2021-2022 учебном году. По причине большого объема необходимой работы он не приступил к написанию алгоритма, но занимался анализом Европейской части РФ и снижением объема задачи. После этого он столкнулся с такой проблемой как невозможность получения математически точных данных для составления кольцевых маршрутов. Работу Фролова нельзя охарактеризовать завершенным продуктом, но на основе его деятельности можно сделать выводы об интересной для туристов территории России, плотности железных дорог и узловых станций на Европейской части России. Результаты работы Фролова отражены в QGIS’е, представлены ниже. Точками типа effect drop shadow (звездочками) изображены узловые станции, а красными точками – исторические поселения.



# Решение

## Теоретические выкладки

В алгоритме используется достаточно простая геометрия: теорема Пифагора, скалярное произведение векторов, а также базовые тригонометрические функции. Теорема Пифагора нужна для нахождения расстояний: длины вектора, ближайшего расстояния между станцией из списка от РЖД и точкой из изображенных в QGIS’е данных (таким образом два файла данных соотнесены). Скалярное произведение же нужно, поскольку во многом алгоритм основан на выборе вектора с углом (к вертикали) самым близким к идеальному, для этого же используются тригонометрические функции.

## Алгоритмы

Существует множество алгоритмов, которые заключаются в нахождении колец (замкнутых графов) на основе матрицы смежности данного графа: алгоритм объединения-поиска, алгоритмы, основанные на последовательном поиске в глубину и поиске в ширину. Эти алгоритмы основываются на теории графов, ведущей свое начало от Леонарда Эйлера и его решения задачи о семи кенигсбергских мостах (которая в своем роде тоже является задачей о построении кольцевого маршрута). Все они не актуальны для текущей задачи по следующей причине: все эти алгоритмы созданы для неориентированного графа, представленного в абстрактном, математически точном пространстве, то есть алгоритмы не учитывают (и не могут учитывать по своей сути) какие-либо географические составляющие, а также исторические поселения (точки, заход в которые маршрутом является приоритетным). Наглядно поясняет неприменимость подобных алгоритмов для проекта решение задачи о кенигсбергских мостах: сведя задачу к чисто математической, представив кенигсбергские мосты как ребра графа, а части центра города как вершины, теория графов лишила задачу какой-либо ценности с точки зрения туристической. Анализом подобных алгоритмов и их пригодности для задачи построения туристических кольцевых маршрутов занимался еще Александр Фролов: его выводы заключались в том, что “легче писать алгоритм самому”.

В проекте используются обновленные для текущей задачи базовые алгоритмы, в большинстве основанные на алгоритме нахождения экстремума.

## Программная реализация

Начальные данные представлены в двух видах: есть Excel-файлы со списком станций РЖД и со списком исторических городов (официально включенных в перечень Министерства Культуры РФ), а также есть нарисованная в ГИС-приложении QGIS карта железных дорог России.

QGIS – геоинформационная система, удобная для автономной работы с геопространственной информацией, весьма наглядная. В ходе работы над проектом Александра Фролова в QGIS’е появились слой с узловыми станциями РЖД, а также с историческими поселениями, которые соответствуют станциям РЖД (этот слой не был окончательно заполнен, поэтому в ходе моей работы алгоритмически была получена таблица, где были соотнесены железнодорожные станции и исторические поселения).

Excel – всем известная программа для работы с электронными таблицами. Для текущей задачи удобна, поскольку изначально данные от Путевой Части РЖД хранятся в Excel-файле, а также Excel позволяет с помощью фильтров анализировать плотность железнодорожных станций, исторических поселений и столиц субъектов РФ на Европейской части России.

Работа со всеми необходимыми данными осуществляется с помощью системы управления базами данных PostgreSQL: преимущество этой СУБД при работе над данным проектом заключается в том, что она объектно-ориентированная и имеет расширение PostGIS, добавляющее поддержку географических объектов. Таким образом, с помощью этой СУБД можно работать с необходимыми для проекта геометрическими объектами (линия и точка), а также взаимодействовать с QGIS’ом (который был разработан в первую очередь для кроссплатформенной работы с данными PostGIS). С помощью расширения PostGIS была решена проблема, которая не позволила реализовать проект Фролову: данные были извлечены из соответствующего слоя QGIS в таблицу в PostgreSQL, после чего с помощью соответствующий функций были конвертированы в файл с начала и конца отрезка, представляющего с собой две соединенные между собой станции.

Проект написан на языке Python в среде разработки PyCharm: язык является удобным для алгоритмической работы с QGIS’ом, PostgreSQL и Excel. Использовались соответствующие библиотеки: openpyxl – для работы с Excel-файлами, а также psycopg2 – для работы с PostgreSQL. Последняя библиотека позволяет составлять SQL-запросы к базе данных внутри кода, что позволяет значительно упростить синтаксис запросов.

# Ход работы

## Анализ задачи Александром Фроловым

Появилась задача осенью 2021 года, и ей занимался Александр Фролов. В ходе его работы в течение года (которая во многом заключалась в анализе задачи) QGIS-проект с картой железных дорог стал более наглядно отражать плотность исторических поселений и железнодорожных станций одновременно. Также Фролов исследовал ранее существующие алгоритмы со схожими задачами (основанные на теории графов). Помимо этого в это время появились наброски первых колец, соответствующих критериям туристического кольцевого маршрута (колец, необходимых для общего понимания задачи и ее анализа). Сама работа Фролова по созданию алгоритма также была основана прежде всего на теории графов и матрице смежности, но в открытом виде не было соответствующих данных (которые пришлось впоследствии извлекать из QGIS’а с помощью изучения его фундаментальных основ), поэтому его работа не могла быть реализована, и реализация проекта была отложена до следующего года.

## Соотнесение станций и исторических поселений

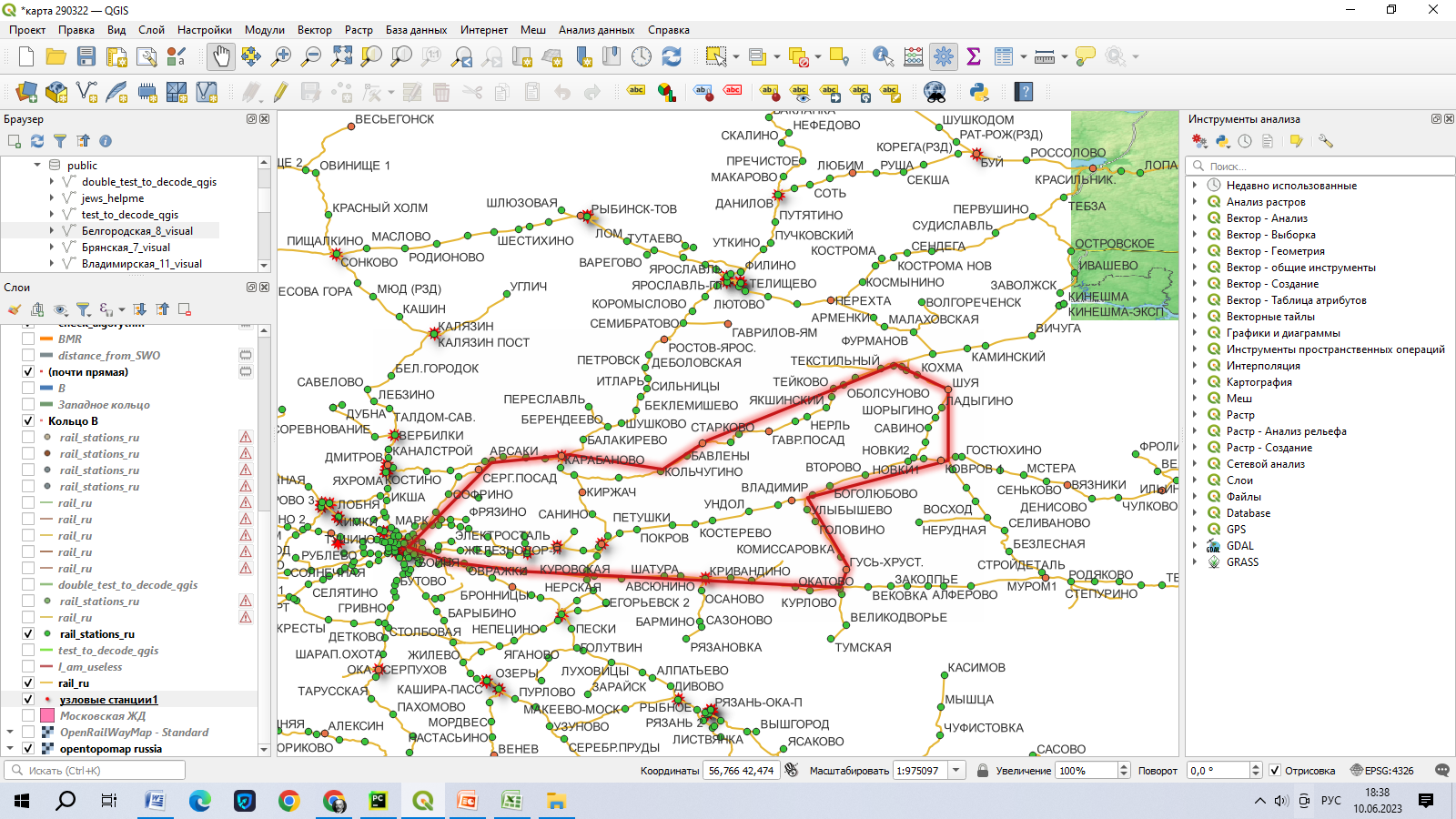
После завершения работы Фролова над проектом возобновился процесс решения данной задачи в ноябре 2022 года. Первой стадией работы было соотнесение исторических поселений со списком станций РЖД от Путевой Части. Для решения этой подзадачи был создан алгоритм, основанный на сопоставлении названий железнодорожных станций и названий исторических поселений. Основной проблемой данного алгоритма были разнообразные варианты сокращений названий поселений, используемых в РЖД. Среди самых оригинальных вариантов сокращений названий станций были: “С-ПЕТЕР-БАЛТ” (Санкт-Петербург-Балтийский), “ГАВР.ПОСАД” (Гаврилов Посад), “НИЖН.НОВГ-МС” (Нижний Новгород-Московский). На основе этих названий можно понять, что хотя название станции скорее всего угадывается по его сокращению, но принципы, по которым работники РЖД сокращают слова, весьма нестабильны: так в роли разделителей между словами может выступать “.”, “-“ или “ “ (например, “ПЕНЗА 1”), части слова могут быть сокращены вплоть до одной буквы (“С-ПЕТЕР-БАЛТ”), что необходимо учесть в алгоритме. Между тем автоматизация процесса была необходима, поскольку число станций, с которыми необходимо работать, превышает 3 тысячи, а на основе предварительной работы Фролова можно было сделать вывод о том, что неавтоматизированная работа с подобным объемом данных ведет к появлению т.н. dirty data (так в слое QGIS “исторические города” отмечены далеко не все те поселения, которые есть в конечных данных проекта).

## Анализ перспектив алгоритмического построения колец

Параллельно проходил анализ предметной области и всего созданного прежде: образцовых кольцевых маршрутов и слоев узловых станций и исторических городов в QGIS’е.

Образцовый кольцевой маршрут, найденный в результате ручного исследования, представляет из себя восточное кольцо от Москвы. Кольцо представлено ниже, визуализированное с помощью QGIS’а. Первоначальная его задумка основана на маршруте от Ярославского вокзала по следующим узловым станциям: МОС-ПАС-ЯРОС – ПОСТ 81 КМ – АЛЕКСАНДРОВ – БЕЛЬКОВО – ИВАНОВО – НОВКИ 1 – ВЛАДИМИР – ОКАТОВО – НЕЧАЕВСКАЯ – КРИВАНДИНО – КУРОВСКАЯ – ЛЮБЕРЦЫ, где маршрут считался завершенным, так как из Москвы пассажиры вновь попадали в Москву. Исторически интересными городами этого маршрута являются: Сергиев Посад, Александров, Юрьев-Польск, Гаврилов Посад, Иваново, Шуя, Владимир, Гусь-Хрустальный, Гжель.

Впоследствии были внесены следующие коррективы: все маршруты от Москвы принято было считать начинающимися и заканчивающимися на Большом кольце Московской железной дороги – кольце вокруг Москвы, используемое сейчас для грузоперевозок и редкими пассажирами как обычный транспорт. Таким образом, обновленный маршрут начинался в Александрове и заканчивался в Куровской. Ниже можно видеть эскиз первого варианта маршрута, нарисованный в QGIS’е.



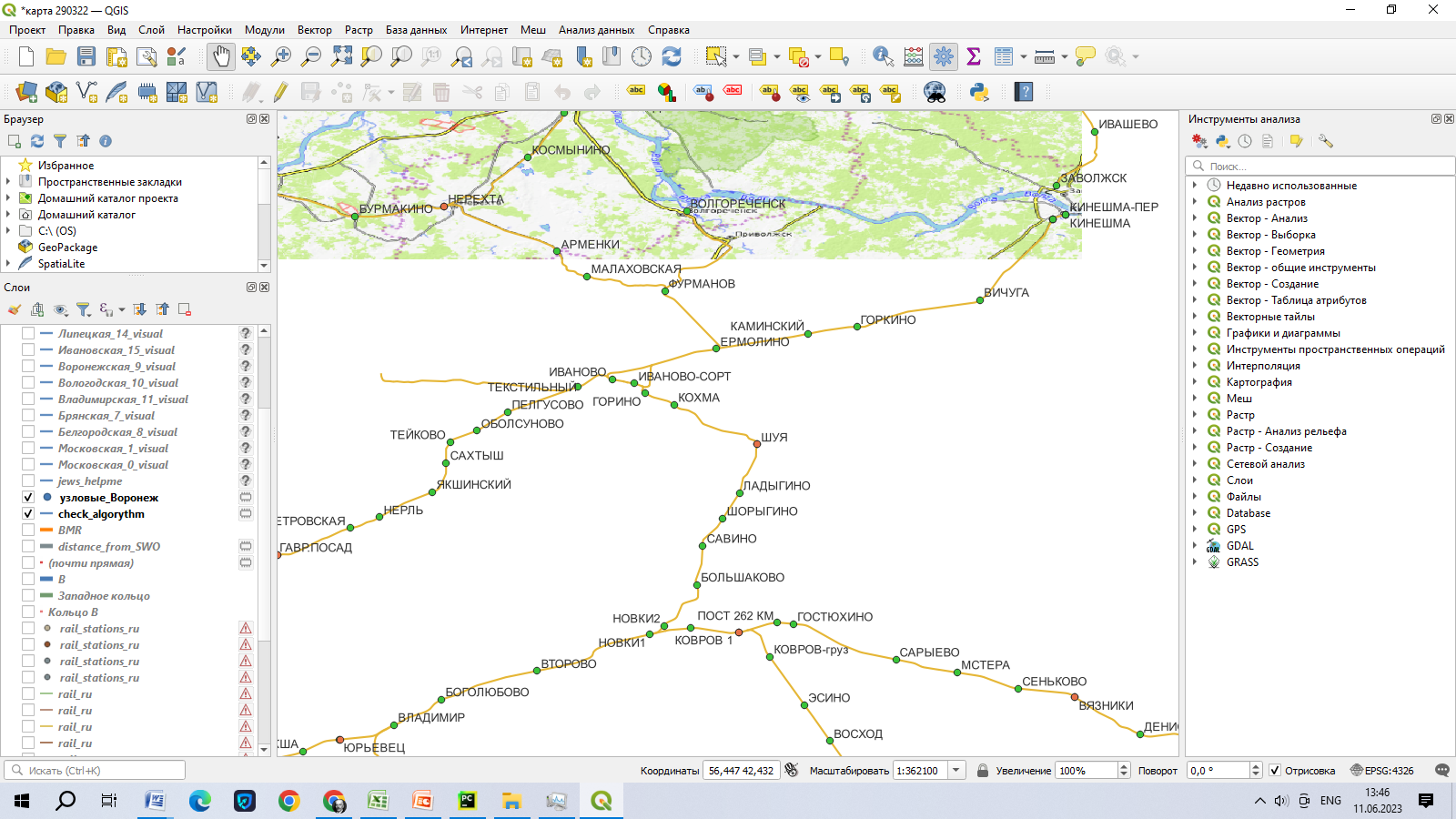
## Процесс формулировку постановки задачи

Вместе с этим формировалось представление об объеме задачи. Первоначально предполагалось, что алгоритм будет рассчитан на нахождение кольцевого маршрута в любом прямоугольнике 3\*4 градуса (числа взяты на основе образцового кольца), а таковые прямоугольники будут перебираться по всей Европейской части России циклом с определенным шагом. Но такая постановка задачи звучала достаточно отдаленно от реальности и приближенно к задаче математической, решаемой в первую очередь через теорию графов и представление графа в векторном пространстве. Поэтому задача требовала переосмысления целевой аудитории: при дополнительном анализе Европейской части России было выяснено, что наиболее актуален проект для жителей столиц субъектов Российской Федерации, поскольку в этих городах банально набирается достаточное число заинтересованных в туристическом маршруте школьников (а проект всегда был рассчитан в первую очередь на школьников). Отсюда вытекает переформулированная постановка задачи: вокруг каждой столицы субъекта Российской Федерации найти по возможности 4 кольца – сторонам света как направлениям. Между тем впоследствии, на основе анализа отдаленных от административного центра страны субъектов РФ выяснилось, что один кольцевой маршрут, соответствующий критериям адекватности туристического кольцевого маршрута, может покрывать все представляющий туристический интерес поселения субъекта. В связи с этим отпала необходимости в построении сразу 4 кольцевых маршрутов вокруг субъекта – постановка задачи свелась к алгоритмическому построению одного кольцевого маршрута на один субъект РФ (в приоритете – чтобы маршрут оставался целиком внутри этого субъекта, хотя это условие, разумеется, труднодостижимо).

## Первая попытка алгоритмического получения данных

Параллельно с анализом задачи и формулированием реалистичной и приближенной к целям туризма, связанного с РЖД, постановки этой задачи были попытки решить алгоритмически проблему, с которой столкнулся Фролов, - проблему нехватки данных. Попыток “алгоритмического решения” было три: все они были основаны на предположении, что в основе карты железных дорог России можно выявить некую математическую закономерность, связанную с географическими координатами станций и расстояниями между станциями.

Первая попытка алгоритмического решения была на расстоянии между станциями и предположении, что все неузловые станции, принадлежащие к одному вектору, стремятся выстроиться в прямую, а изменения направления железной дороги происходят только на узловой станции. Соответственно, алгоритм заключался в следующем: для каждой неузловой станции находили N-ое число ближайших к ней станций (в ходе работы над алгоритмом в качестве числа N последовательно были перебраны числа от 5 до 15), после чего среди этих станций ищутся такие две, чтобы расстояние от прямой (которую эти две станции образуют) до данной неузловой станции было наименьшим. Таким образом алгоритм находил для каждой неузловой станции такие две из N-ого числа соседних, чтобы все 3 вместе образовывали бы ломаную, которая больше всего походила бы на прямую. При запуске этого алгоритма и анализе полученных данных (которые довольно часто были неудовлетворительны) было выяснено, что предположение ошибочно: примером, опровергающим данное предположение, может служить станция ШУЯ, которая не является узловой (соединена всего с двумя станциями), но при этом на ней направление железной дороги меняется больше, чем на 90 градусов. Ниже можно увидеть данный пример:



Также впоследствии было выяснено, что система, по которой железнодорожная станция получает статус узловой, не соответствует определению узловой станции из теории графов: гораздо больше станций России являются узловыми точками графа (если представлять железнодорожную сеть России как граф), чем имеют статус узловой станции в списке от Путевой Части – эта информация была получена уже после получения достаточно точных данных о железнодорожной сети России с помощью QGIS’а.

Таким образом, первый алгоритм был выявлен как тупиковая ветвь и заброшен (к началу марта 2023 года).

## Вторая попытка алгоритмического получения данных

Второй алгоритм был основан на другом тезисе: от каждой узловой станции отходит не более 4 направлений (это правда: если от города отходит более 4 направлений, то в городе находится более одной узловой станции) – эти направления можно разделить по сторонам света. Соответственно, алгоритм заключался в проложении от каждой узловой станции 4 условных векторов с направлениями строго по сторонам света и соотнесении ближайших неузловых станций с одним из этих 4 векторов. При этом во втором алгоритме учитывалась возможность смены направления внутри одного вектора. Таким образом, алгоритм был направлен на создание матрицы смежности графа в векторном пространстве на основе направлений векторов. В результате запуска алгоритма и анализе полученных данных был сделан вывод о том, что предположение о соотнесении векторов от узловой станции с направлениями света ошибочно. Также на неудовлетворительный (однако лучший, чем в первом алгоритме) результат повлияли описанные выше различия между статусом узловой станции в списке Путевой Части и в теории графов. Второй алгоритм был объявлен тупиковой ветвью в конце марта 2023 года.

## Попытка построения математической модели для получения данных

В ходе анализа результатов обоих алгоритмов был сделан следующий вывод: нельзя применять одну строгую математическую закономерность ко всей железнодорожной сети Европейской части России – во многом связи между станциями зависят от географического положения станций и исторического контекста этого географического положения. В связи с этим возникла идея о создании более сложной математической модели, в которую укладывался бы граф железнодорожной сети России, на основе алгоритмов, предназначенных для задач классификации. Адаптируя задачу получения данных под данный тип задач: для каждых двух узловых станций и третьей неузловой данной определить, принадлежит ли данная неузловая станция вектору из двух узловых станций. Для создания математической модели для решения данной задачи была вручную составлена матрица смежности, которая впоследствии алгоритмически была конвертирована в удобный для решения задачи вид: список аргументов и результат (1 или 0) для каждой тройки – неузловая станция и две узловые. В качестве аргументов были взяты: расстояние от неузловой станции до вектора, расстояние по широте и долготе от каждой из двух узловых станций. Логика, по которой выбрали данные аргументы, основано на предположении, что разница в широте и долготе у двух смежных узловых станций математически связана и эту связь можно выявить алгоритмически с помощью достаточно большого объема данных с разными географическими положениями, представленного в виде матрицы смежности. Эта гипотеза учитывала в том числе и проблемы предыдущих двух алгоритмов: при получении данных о графе железнодорожной сети невозможно использовать выверенные вручную константы, поскольку в зависимости от географического положения константы будут разниться. Разница в широте и разница в долготе же должны были определять эти константы, минуя необходимость определить принадлежность станций конкретной области.

В ходе работы над данным способом получения данных было перепробовано большое количество различных алгоритмов, предназначенных для задач классификации. Это связано с двумя очевидными (после нового, более тщательного анализа железнодорожной сети) ошибками. Во-первых, все три вышеописанных алгоритмических способа получения данных основаны на списке станций от Путевой Части, стало быть, изначально значительно теряли в точности из-за особенностей присвоения станции статуса узловой, указанных выше. Во-вторых, этот конкретный способ заключался в попытке найти сложную математическую зависимость проложения железных дорог от географических координат, но принимал исторический фактор за оказавший на проложение железных дорог недостаточно большое влияние. Между тем во многом при проложении железных дорог исторически учитывали не столько географические координаты и даже не столько географическое положение, сколько насущные экономические интересы: долгое время, вплоть до правления Александра III, процесс строительства железных дорог осуществлялся под руководством различных частных предпринимателей. Соответственно, строительство железных дорог чрезвычайно значительно зависело от ежеминутных интересов капиталистического класса. Когда правительство вернуло себе монополию на руководство строительством железных дорог, оно вновь стало действовать из чисто экономических интересов: проложили практически пустынные ветки дорог на север для добычи руд. Из-за невозможности учитывать вышеописанный исторический фактор в качестве аргумента для построения модели математической зависимости третий способ оказался тупиковой ветвью, в результате чего работа над ним прекратилась в начале мая 2023 года.

## Расширение информации о железнодорожных станциях

Вместе с тем в результате выяснения о несоответствии данных от Путевой Части математически точной теории графов (в угоду более актуальным для железнодорожных станций критериям) возникла потребность в расширении имеющихся данных о железнодорожных станциях Европейской части России. Таким образом, была простейшим парсингом страницы <https://dic.academic.ru/dic.nsf/ruwiki/433064> (список узловых станций в Российской Федерации) расширена информация о железнодорожных станциях списка Путевой Части до достаточно точного списка станций с достаточно точными указаниями узловых станций. Между тем, несмотря на расширенный набор данных, он все равно не являлся абсолютно точным с точки зрения теории графов. Во многом это было обоснованно следующим: узловые вершины железнодорожного графа, не получившие статуса узловой станции, имеют достаточно неперспективные ответвления. Во многом таковые ответвления находятся в достаточно сомнительном благосостоянии, станции, лежащие на таких ответвлениях, часто являются деревнями. Ярким примером может служить станция Соблаго: несмотря на 3 направления, которые от нее отходят, одно из весьма сомнительно – направление Соблаго-Торжок. Помимо достаточно болотистой местности, при тщательном изучении данного ответвления мы можем обнаружить, что большинство станций, принадлежащих этому вектору, заброшены по историческому фактору. Данная ветка официально признана неперспективной: и не в последнюю очередь из-за таких веток информация о графе железнодорожной сети с точки зрения математической теории может не совпадать с информацией от Путевой Части.

## Получение данных с помощью геокодирования

Вследствие невозможности каким-либо образом математически добавить в качестве начальных данных для алгоритмического нахождения связей в графе железнодорожной сети исторический фактор, серьезно повлиявший на проложение железных дорог, были начаты поиски получения данных с помощью геокодирования. В результате изучения прикладного к QGIS’у и PostGIS’у геокодирования была выявлена система, в которой закодированы данные железнодорожной сети. Для упрощения работы железнодорожный граф был представлен в виде ломаной (было значительно упрощено отображение того, как “извивается” железная дорога). После этого с помощью возможностей QGIS’а данная ломаная была представлена в виде списка объектов типа linestring – объектов, используемых в PostgreSQL как геометрические объекты для представления отрезка. Затем каждый объект такого типа был разбит с помощью встроенных функций PostgreSQL на два объекта: начальную и конечную точку отрезка – объекты типа Point. После были найдены внутренние функции PostgreSQL для извлечения из объектов геометрического типа координат в виде дробных чисел. Таким образом, все полученные данные были извлечены в виде таблицы на более 5 тысяч строк, включающей в себя информацию о координатах начальной и конечной станций звена ломаной, и перенесены в Excel-файл. Использование для хранения этой таблицы Excel’а обусловлено тем, что данная информация является постоянной, а ее обновление в любом случае будет связано с аналогичным вышеописанному процессом получения данных с помощью геокодирования.

## Создание базы данных

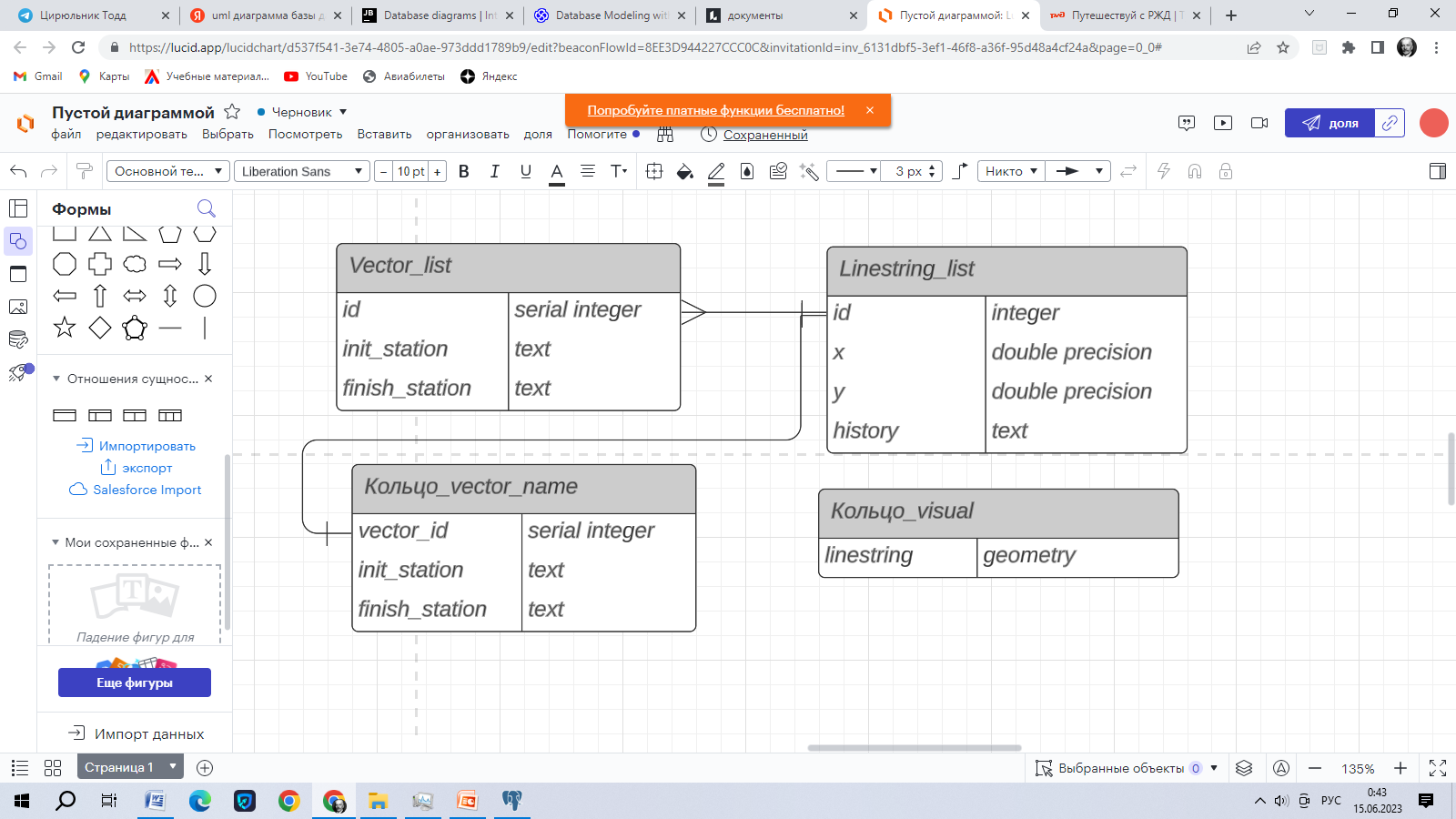
Затем был написан алгоритм, который полученные выше данные о ломаной железнодорожной сети, представляет в виде, более соответствующем теории графов. Были выявлены узловые вершины графа железных дорог.

Затем для каждой такой вершины был создан список смежности с дополнительной информацией: для каждого объекта списка смежности в качестве необходимой для векторного представления информации была добавлена информация о звеньях ломаной, соответствующей данному вектору. Данный блок информации был представлен в СУБД PostgreSQL, поскольку эта информация имеет систему, реализуемую с помощью СУБД. Также, забегая вперед, дальнейшие полученные кольца представлены в виде списка векторов, каждый из которых ссылается на соответствующий вектор в базе данных.

Таким образом, была выстроена структура базы данных на основе достаточно логичной части теории графов, а также необходимости представления теории графов в векторном пространстве. Можно сказать, что таблица со списком векторов основана на теории графов, а дополняющая ее таблица – на векторном представлении.

Таким образом, снижена сложность математической составляющей задачи в пользу прикладной. Это актуально для данного проекта, поскольку основываться решение задачи должно на прикладных характеристиках железных дорог.

Визуализируя вышесказанное, мы можем получить следующую схему:



Мы можем видеть две начальные таблицы: vector\_list – список векторов Европейской части России, linestring\_list – список точек, принадлежащих данным векторам. Говоря о начальных таблицах, надо сказать о том, что за счет списка станций от Путевой Части и описанного выше процесса соотнесения этого списка со списком исторических поселений мы можем добавить в базу данных поле history. Данное поле принимает значение – название исторического поселения – если данная точка является ближайшей по координатам к этому историческому поселению. За счет предварительной работе по соотнесению станций и поселений точность этих данных весьма высока. Также мы можем видеть структуру хранения данных полученных алгоритмически колец. Они хранятся в двух видах: виде описательном – Кольцо\_vector\_name, а также виде геокодированном – Кольцо\_visual. Первый вид – представление кольца в виде списка векторов. Этот вид удобен для анализа кольца вручную для определения каких-либо характеристик, присущих железнодорожному маршруту, но трудных для выявления их алгоритмически. Второй вид основан на типе geometry – типе данных PostgreSQL, который поддерживается QGIS’ом. Эта вторая таблица состоит из представленных в качестве типа geometry звеньев ломаной, где ломаная – кольцевой маршрут.

## Попытка создания математической модели построения кольцевых маршрутов

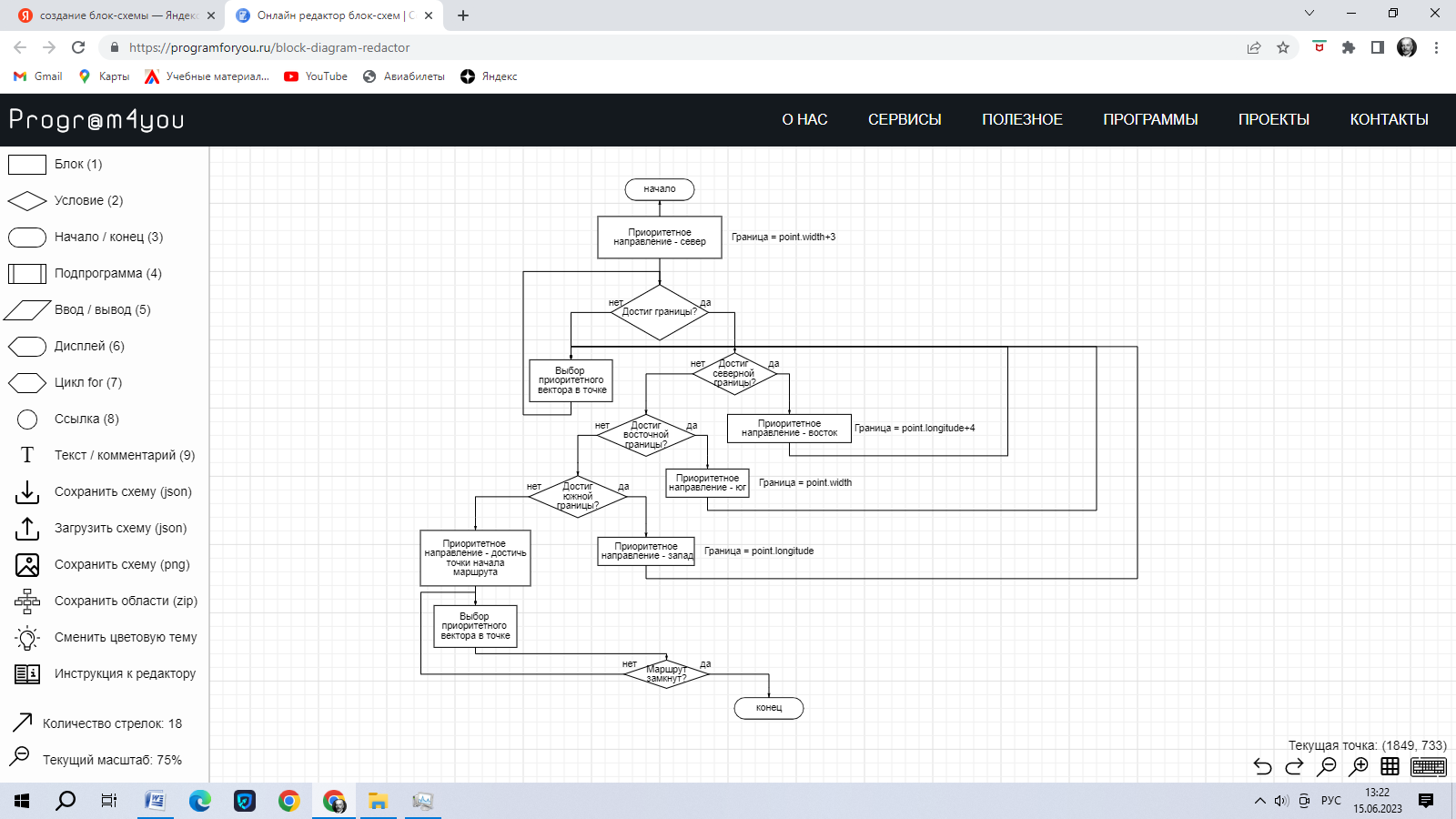
Таким образом, были получены и представлены в удобном формате необходимые данные. Теперь можно было работать с графом железнодорожной сети, претендуя на математическую точность алгоритмов.

Возникло предположение о возможности создания математической модели для создания туристических кольцевых маршрутов. В качестве аргументов для подобной математической модели по задумке можно было использовать число исторических городов в векторе, направление вектора (угол между вектором и вертикалью), а также географические координаты вектора по отношению к начальной точке маршрута.

Таким образом, предполагаемая модель должна была работать для создания кольцевых маршрутов достаточно фиксированной длины и ширины. В данном случае: предполагалось, что идеальный кольцевой маршрут укладывается в прямоугольник 3\*4 градуса. Также этот алгоритм ориентировался на создание 4 кольцевых маршрутов для каждого субъекта РФ – по сторонам света.

Соответственно, при написании алгоритма для данной модели был принцип, основанный на том, что при начале создания кольца задается приоритетное направление: север или юг. Затем математическая модель выбирает на основе образцовых колец в качестве начальных данных лучший вектор от данной точки. Затем, когда маршрут таким образом добирается до границы прямоугольника, приоритетное направление меняется. Так происходит 4 раза, в результате чего по задумке алгоритма должно быть выстроено кольцо с достаточно четкими разницами в долготе и широте.

Ниже визуализирован алгоритм поиска кольцевого маршрута по направлениям север-восток-юг-запад. Алгоритмически был реализован цикл, в котором были перебраны все 4 возможных направлений (где на первом месте обязательно север/юг).



Для создания математической модели вручную были проанализированы некоторые туристические маршруты, соответствующие описанным в начале критериям. Но в ходе запуска данного алгоритма было выявлено, что данных, полученных таким способом, недостаточно для создания адекватной математической зависимости. В частности было обнаружено, что аргумент, отвечающий за направление вектора, используется моделью сильно слабее, чем должен.

К тому же, проанализировав Европейскую часть России, было выявлено несколько замечаний, которые корректируют изначальный подход к алгоритмическому построению колец.

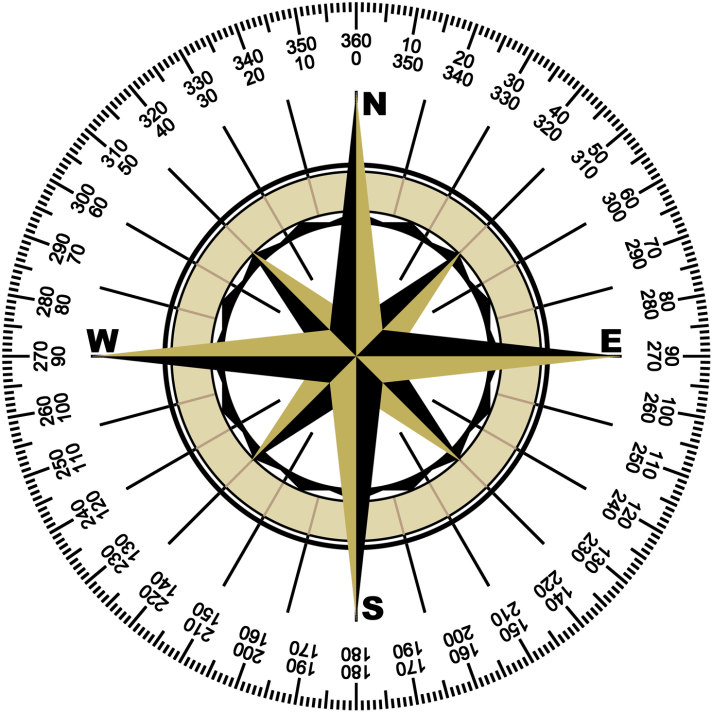
Во-первых, зачастую кольцевой маршрут не укладывается в рамки прямоугольника 3\*4: это зависит от устройства железных дорог в данной области. Например, южнее Московской области дороги устроены таким образом, что возможно создание кольцевых маршрутов, достаточно коротких и при этом интересных с точки зрения туризма. Подобные маршруты в каких-то случаях оказываются меньше по размерам, чем прямоугольник 3\*4. А севернее дороги одинокие: зачастую железнодорожная сеть там состоит из длинных векторов – направлений в одну сторону. Такие векторы встречаются в районах Вологодской, Нижегородской областях, а также севернее. Таким образом, применение каких-либо констант для создания четкой модели кольцевого туристического маршрута в данном проекте неуместно.

Это было выявлено в результате запуска описанного выше алгоритма, в результате чего идея о создании математической модели для построения кольцевых маршрутов была заброшена.

## Алгоритм построения кольцевых маршрутов

После дополнительного анализа Европейской части России и отобранных вручную колец было в качестве самой разумной основы для построения колец выявлено построение на основе исторических поселений субъектов РФ. Была обновлена база данных: было добавлено соотнесение субъектов РФ и их исторических городов.

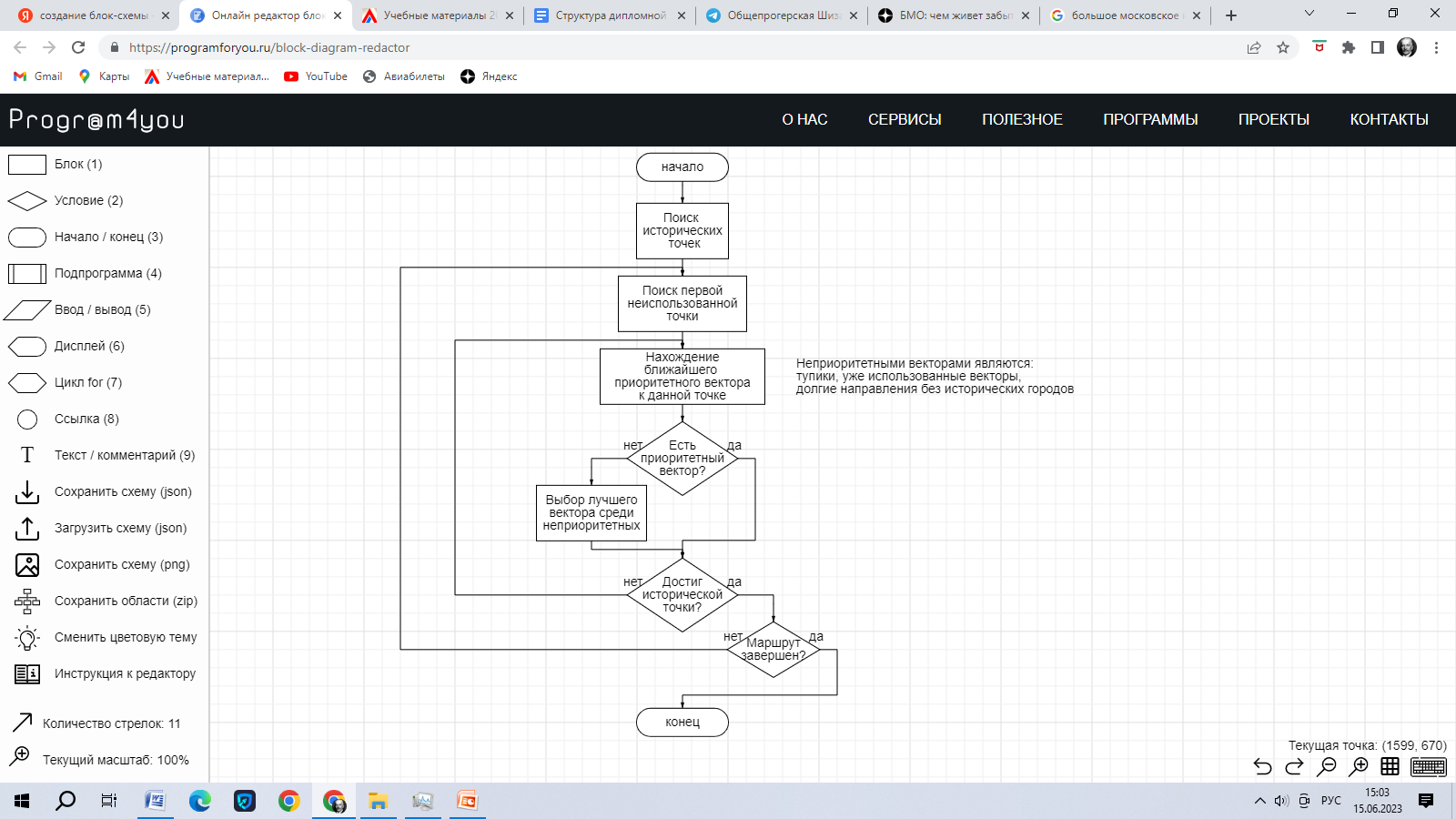
Принцип алгоритма заключается в так называемом “движении по часовой стрелки”. Суть его в том, что первоначально создается массив приоритетных для посещения маршрутом точек – исторически интересных железнодорожных станций в данном субъекте РФ. Затем алгоритм находит ту, направления к которой от начальной точки маршрута (столицы субъекта РФ) задано минимальным углом. В данном случае угол направления считается с помощью очевидного инструмента геокодирования – компаса:



После выявления первой такой точки алгоритм ищет последовательно векторы, направление которых стремится к данной точке. При этом, разумеется, исключаются из алгоритма неинтересные с точки зрения туризма тупики. Таким образом, алгоритм позволяет найти кратчайший путь до точки. При этом в качестве предметной области используется в гораздо меньшей степени теория графов, чем векторное представление и геокодирование. Это связано с тем, что использование в данной задачи теории графов, хотя и будет более математически точно, но неоптимизированно и имеет высокую алгоритмическую сложность.

Вышеописанный алгоритм повторяется для каждой исторической точки субъекта, причем конечной точкой является столица субъекта (начальная точка). Таким образом, маршрут оказывается замкнутым и затрагивает все возможные исторические местности данного субъекта РФ. Разумеется, в алгоритме прописаны исключения на случай, если кольцевой маршрут выходит слишком короткий или слишком продолжительный.

Ниже мы можем видеть данный алгоритм представленный в виде блок-схемы (без уточняющих конструкций if-else для проверки, не является ли маршрут слишком коротким/долгим):



Также в алгоритме прописано особое исключение для Москвы. Неудивительно, что Москва является субъектом с самой разветвленной сетью железных дорог в России. Но для данной задачи было бы бесполезно и сложно реализовать маршрут от конкретного московского вокзала до него же. Для начальной и конечной точки маршрута было выбрано Большое Московское кольцо, уже упоминаемое в описании образцового кольцевого маршрута. Попадание на одну из точек этого кольца было признано для маршрутов, отходящих от Москвы, завершением маршрута. Ниже можно видеть данное кольцо:



После долгого редактирования алгоритм был запущен, в результате чего было получено 16 кольцевых маршрутов на Европейской части России.

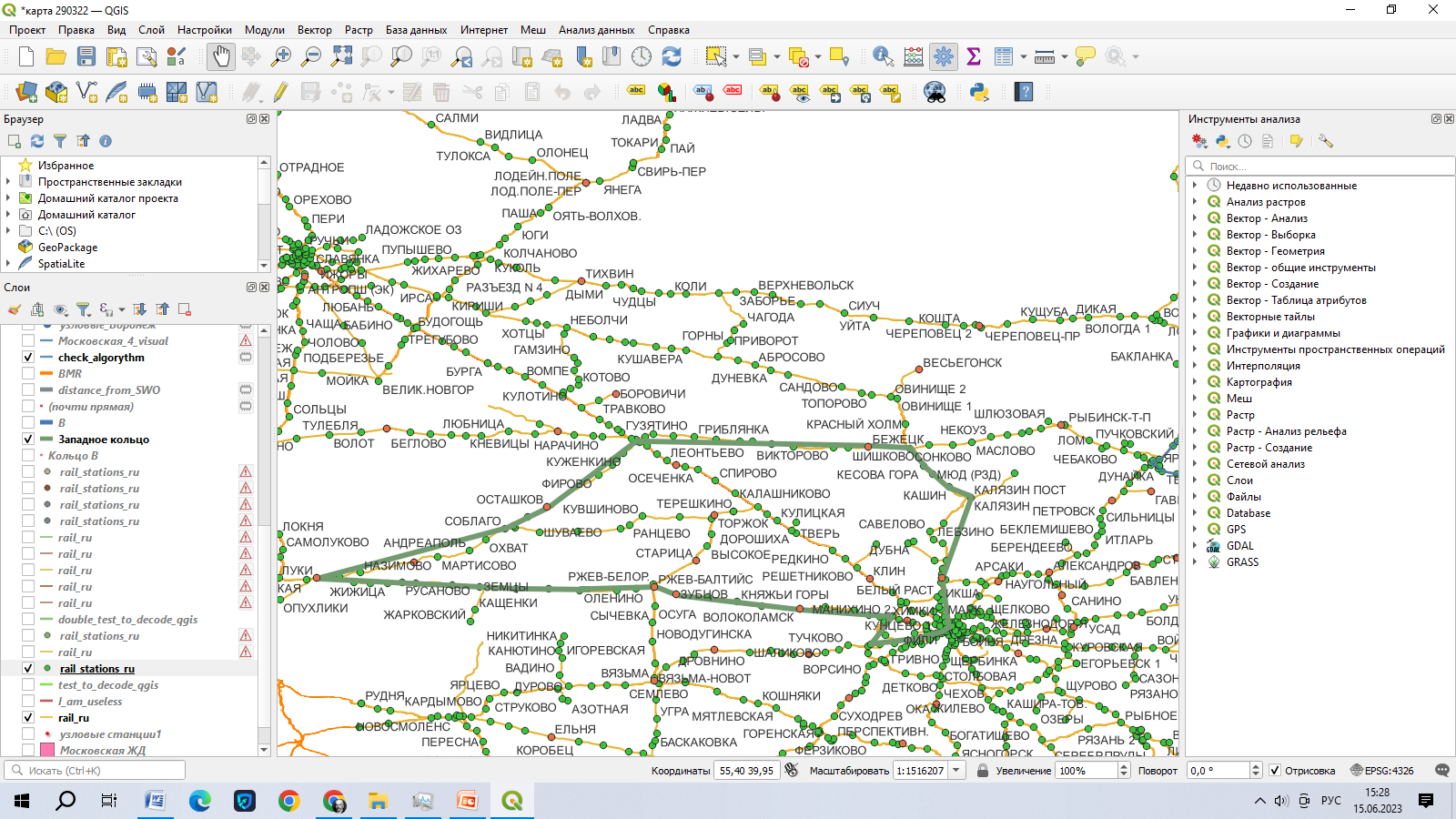
## Визуализация

Полученные кольца нуждались в визуализации в ГИС-приложении. Поскольку инструменты геокодирования QGIS’а были уже изучены в ходе работы над проектом, данная подзадача не составила большого труда. Возвращаясь к структуре базы данных, мы можем видеть, что таблицы типа “Кольцо\_visual” предназначены в первую очередь для визуализации в QGIS’е. Таким образом, в код просто были добавлены соответствующие sql-запросы типа “create” и “insert”. В результате в QGIS’е визуализированы все существующие полученные кольца.

# Результат

## Решение поставленной задачи

Алгоритм, основанный на движении “по часовой стрелки” был реализован полностью на всей Европейской части России. Однако очевидно, что данный алгоритм не охватывает весь потенциал кольцевых туристических маршрутов. Так в ходе анализа туристических кольцевых маршрутов был найден следующий кольцевой маршрут от Москвы:



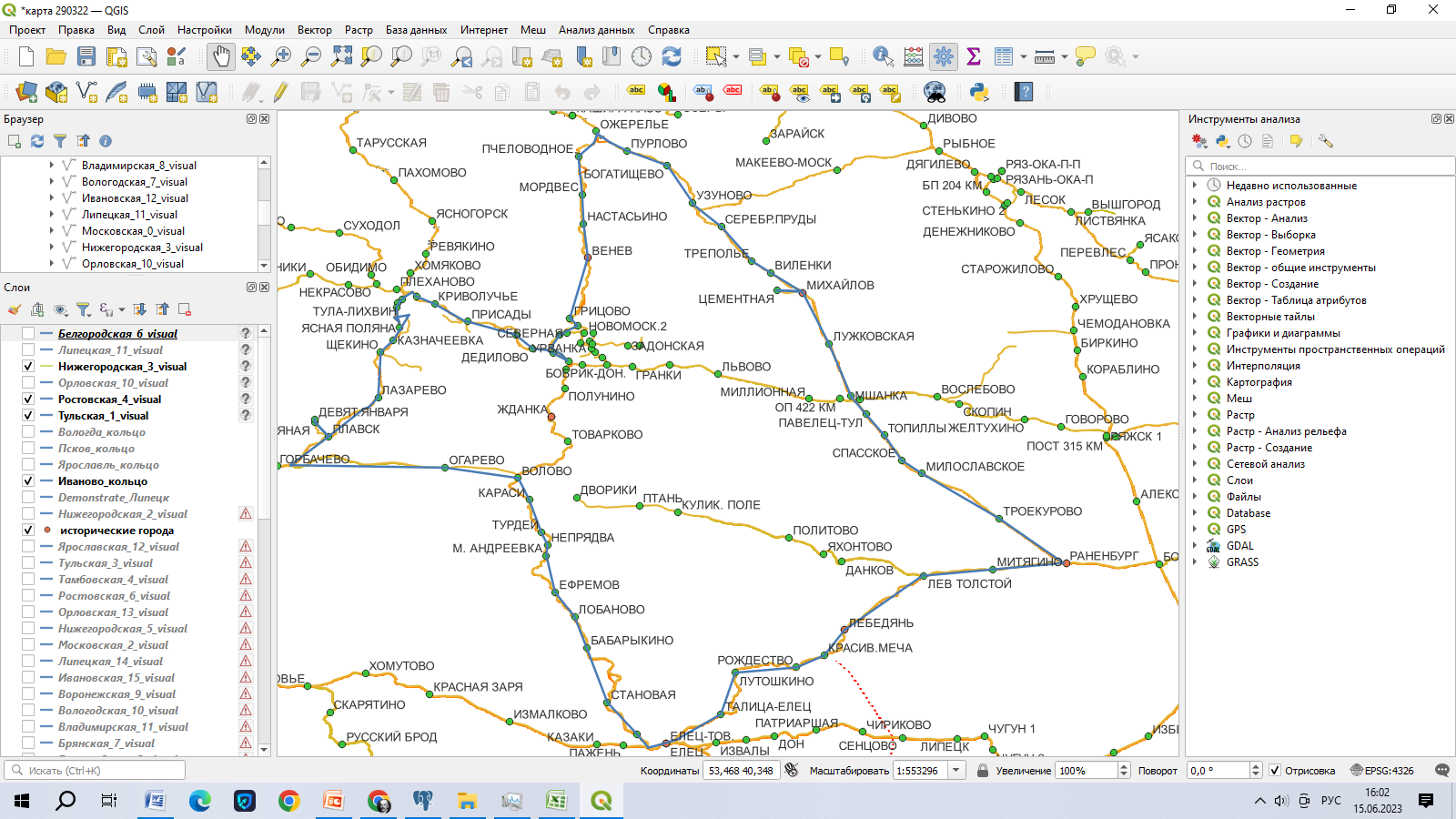
Между тем в результате исследования полученных алгоритмически кольцевых маршрутов было выявлено, что для изображенного выше кольцевого маршрута алгоритм неактуален. Это связано с тем, что исторические города данного маршрута не лежат Москве и даже в окружающей Москву зоне. Таким образом, для получения алгоритмически этого маршрута и аналогичных нужен другой алгоритм.

Также для расширения объема полученных данных возможно добавление противоположного алгоритма – движения “против часовой стрелки”. Так для некоторых субъектов РФ данное дополнение было бы важным, поскольку зачастую интересные для туристов железнодорожные направления являются взаимоисключающими при построении кольцевого маршрута. А при построении сразу двух кольцевых маршрутов по противоположным алгоритмам было бы возможно охватить сразу весь туристический потенциал субъекта.

## Исследование полученных колец

Говоря о полученном в ходе запуска алгоритма результате, в первую очередь нужно сказать о кольцевых маршрутах области южнее Москвы и западнее Волги. Поскольку в данной части России высокая плотность железных дорог, разветвленная сеть, а также большое число исторических городов в каждом субъекте, кольца, полученные в этих субъектах, являются одними из более соответствующих критериям образцового туристического кольцевого маршрута. Эти маршруты по продолжительности укладываются в рамки оптимальных маршрутов, а число исторических местностей, затрагиваемых этими маршрутами, достаточно высоко. Можем рассмотреть несколько маршрутов из этой части России подробно.

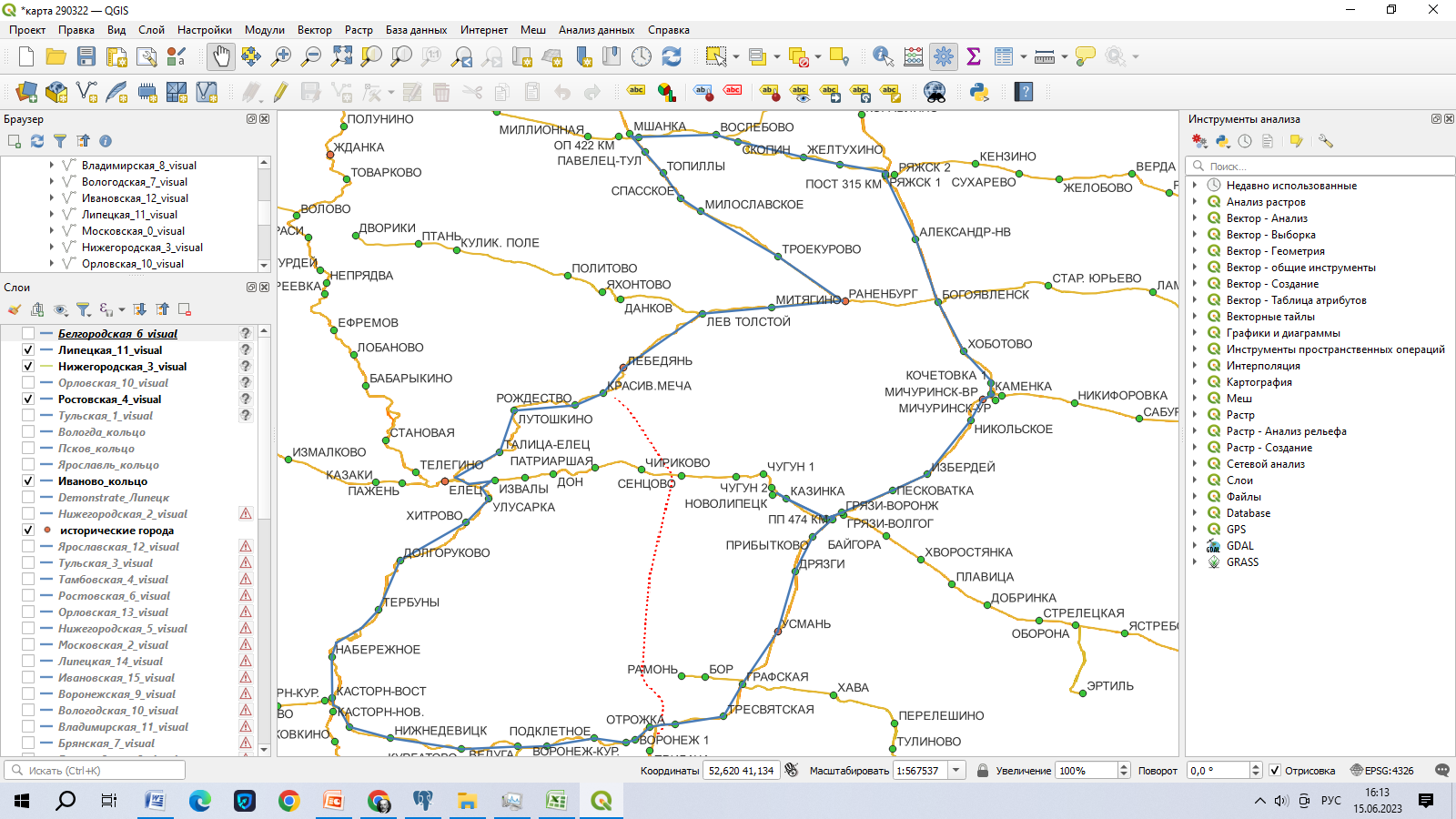
Так выглядит кольцо в Тульской области:



Анализируя его, можно сказать, что маршрут затрагивает три исторические станции в Тульской области: Тулу, Венёв и Плавск – а также 4 исторические станции из ближайших к нему субъектов: Михайлов, Раненбург, Лебедянь, Елец. Помимо исторических городов из официального перечня, этот маршрут затрагивает две интересные и уже упоминаемые станции: Ясную Поляну и Лев Толстой. Таким образом, этот маршрут выглядит весьма солидно.

Также именно на основе его анализа был сделан вывод о возможности добавления противоположного алгоритма. Дело в том, что большинство исторических городов Тульской области лежит на направлениях, отходящих непосредственно от Тулы. По элементарной теории графов при построении одного кольцевого маршрута можно затронуть всего 2 таких города. При этом 2 не затронутых данным кольцом исторических местности лежат западнее, то есть к ним возможно получить доступ только в том случае, если начинать построение маршрута по принципу, противоположному данному.

Другим интересным кольцом из этой части России является кольцо в Липецкой области:



Как мы можем видеть, оно затрагивает такие исторические местности как Воронеж, Елец Лебедянь, Раненбург, Богоявленск, Мичуринск, а также множество представляющих культурный интерес без официального Перечня станций.

Подобные кольца есть в этой части России и в области, более близкой к сомнительным геополитическим событиям. Несмотря на нынешнюю обстановку, эти области представляют высокий интерес с точки зрения культуры и истории.

Если же рассматривать кольца из областей севернее, то там зачастую встречаются долгие продолжительные направления в одну сторону. Это связано с особенностями развития железных дорог на севере страны. Между тем кольца на севере от Москвы также не лишены перспективы с точки зрения туристических маршрутов. Там затрагиваются такие интересные с точки зрения туризма города как Вологда, Буй, Галич, Котельнич и прочие. Продолжительность этих маршрутов значительно выше, однако в большинстве она все еще оптимальна. Таким образом, это направление железнодорожного туризма также можно считать перспективным с точки зрения кольцевой модели маршрутов.

Таким образом, анализируя полученные кольца, мы можем сказать, что они достаточно перспективны в плане развития их как полноценные туристические маршруты.

# Заключение

## Перспективы дальнейшей разработки

Полученные в проекте результаты позволяют в дальнейшем перейти к Комплексной оценке ситуации в субъектах Федерации для подготовки проектов новых маршрутов. Необходимо, в частности, учитывать климатические особенности зон прокладки кольцевых маршрутов, наличие или необходимость дополнительного обустройства стояночных территорий, социокультурный уровень посещаемых местностей и т.д.

В ходе поиска партнеров по развитию проекта целесообразно обратиться как к государственным организациям инфраструктурного уровня, так и к ВУЗам различной профессиональной направленности. Возможно также создание общественных объединений, участие в которых смогут принять как студенты и школьники. Так и активисты в сфере краеведения.

Наконец, развитие функционала представленной программы представляется оправданным в направлении совершенствования пользовательского интерфейса и расширения числа источников первичных данных для комплексной оценки зон прокладки кольцевых маршрутов.

## Дальнейшая актуальность темы

Глобально алгоритмическое построение кольцевых туристических маршрутов все еще имеет вполне широкую перспективу развития. Помимо интегрирования данных маршрутов с прочими железнодорожными характеристиками, есть указанные выше возможности для расширения алгоритмов поиска колец.

Говоря в общем, разумеется, железнодорожный туризм все еще актуален для России. Аудитория для подобных маршрутов достаточно нова, перспективна для освоения, а также весьма обширна. Заинтересованного в истории и культуры страны населения достаточно, чтобы все полученные в перспективе развития этого проекта маршруты были актуальны.

# Список литературы

А.В. Омельченко “Теория графов”/Москва Издательство МЦНМО, 2018г. – 412 стр.

О.В.Королева “География туризма: учебно-географическое пособие для бакалавриата по направлению “туризм”/Саратовский Государственный Университет имени Чернышевского, 2014г. – 63 стр.

“Путешествуй с РЖД|Туристические поезда” - <https://www.rzd.ru/ru/10419>

Требушкова И.Е., Полякова Н.О. “Географический анализ государственной политики на железнодорожном транспорте России”/Курский Государственный Университет, 2021г. – 21 стр.

Косарева Н.В., Адашова Т.А. “Железнодорожный туризм как один из векторов устойчивого развития регионов России”/Вестник Алтайской академии экономики и права, 2021г. – 65 стр.

Железнодорожный транспорт. Энциклопедия: Большая Российская энциклопедия, 1995г. – 324 стр.

“БМО: чем живет забытое железнодорожное кольцо” - <https://dzen.ru/media/strelkamag.com/bmo-chem-jivet-zabytoe-jeleznodorojnoe-kolco-5a13ec3d799d9d084f2b6580>

“Ralisa stat UIC” - <https://uic-stats.uic.org/gis/> всемирный атлас ЖД

“Онлайн справочных грузовых Ж/Д станций” - <https://www.alta.ru/railway/>

**Исходный код**

Код, решающий первую подзадачу - соотнесение станций и исторических поселений:

import openpyxl

class Station:

name = ''

longitude = 0 #East-West

width = 0 #North-South

hub = False

history = None

year = None

def \_\_init\_\_(self, name, longitude, width, hub, history ):

self.name = name

self.longitude = longitude

self.width = width

self.hub = hub

self.history = history

def output (self):

result = self.name+' '+str(self.longitude)+' '+str(self.width)+' '

if self.hub:

result+= 'узловая'

if self.history!=None:

result+=' '+self.history

print (result)

def isEqual(self, obj2): #в списке ПЧ встречались продублированные станции

return (self.name == obj2.name and self.longitude == obj2.longitude and self.width == obj2.width and self.hub == obj2.hub)

class History\_towns:

name = ''

year = None

def \_\_init\_\_ (self, name, year):

self.name = name

self.year = year

def compare\_strings(history\_name, station\_name): #основная функция для сравнения названий исторического поселения и станции

delete\_num = 0

for i in range (len(station\_name)):

i = i-delete\_num

try:

temp = int(station\_name[i])

delete\_num +=1

station\_name = station\_name[:i-1]+station\_name[i+1:]

except:

continue

for i in range (len(history\_name)):

if history\_name[i] == 'ё':

history\_name = history\_name[:i]+'е'+history\_name[i+1:]

temp\_history\_name\_list = history\_name.lower().split(' ')#каждая часть названия исторического города может быть сокращена

temp\_station\_name\_list = station\_name.lower().split('.')#в качестве распространенного знака препинания для сокращения используется "."

history\_name\_list = []

station\_name\_list = []

if len(temp\_history\_name\_list)!=len(temp\_station\_name\_list):

return False

for x in temp\_history\_name\_list:

temp\_history\_list\_kill\_me\_pls = x.split('-') #также для сокращения используется "-"

for y in temp\_history\_list\_kill\_me\_pls:

history\_name\_list.append(y)

for x in temp\_station\_name\_list:

temp\_station\_list\_kill\_me\_pls = x.split('-')

for y in temp\_station\_list\_kill\_me\_pls:

station\_name\_list.append(y)

if abs(len(history\_name\_list)-len(station\_name\_list))>1:

return False

for i in range (len(history\_name\_list)):

history\_name\_temp = history\_name\_list[i]

try:

station\_name\_temp = station\_name\_list[i]

if station\_name\_temp != history\_name\_temp[:len(station\_name\_temp)]:

return False

except:

break

return True

if \_\_name\_\_ == '\_\_main\_\_':

result\_station = openpyxl.load\_workbook('result.xlsx')

years\_hist = openpyxl.load\_workbook('history\_sort.xlsx')

result\_station = result\_station['Sheet']

years\_hist = years\_hist['Лист1']

hist\_list = []

list\_station = []

for row in result\_station:

hub = False

if row[0].value!= 'name' and row[0].value!= None:

hub = (row[3].value!=None)

list\_station.append(Station(row[0].value, row[1].value, row[2].value, hub, row[4].value))

for row in years\_hist:

if row[3].value!=None:

hist\_list.append(History\_towns(row[3].value, row[6].value))

for x in hist\_list:

for y in list\_station:

if compare\_strings(x.history, y.name): #сравнение названий станции и исторического города

y.year = x.year

break

wb = openpyxl.Workbook()

ws = wb.active

for x in list\_station:

if x.hub == True:

x.hub = 'узловая'

else:

x.hub = None

ws.append({'A': 'name', 'B': 'долгота', 'C': 'широта', 'D': 'узловая', 'E': 'исторические поселения', 'F': 'год основания'})

for x in list\_station:

ws.append({'A': x.name, 'B': x.longitude, 'C': x.width, 'D': x.hub, 'E': x.history, 'F': x.year})

wb.save ('finally\_ready\_station\_file.xlsx')

Код, парсящий страницу для расширения данных о железнодорожных станциях:

import bs4

import os

import requests

URL\_TEMPLATE = 'https://dic.academic.ru/dic.nsf/ruwiki/433064'

if \_\_name\_\_ == '\_\_main\_\_':

r = requests.get(URL\_TEMPLATE)

print (r.status\_code)

soup = bs4.BeautifulSoup(r.text, 'html.parser')

result = soup.get\_text().split()

list\_result = []

for x in result:

if not x in list\_result:

temp = ''

for y in x:

if y!='(' and y!=')' and y!=',':

temp+=y

list\_result.append(temp)

f = open('result.txt', 'w')

for x in list\_result:

try: f.write(x+'\n')

except: print(x)

f.close()

Код, вносящий эти данные в общий список железнодорожных станций:

import openpyxl

class Station:

def \_\_init\_\_(self, name, longitude, width, hub, history, year, isSubject, isBMR):

self.unified = False

self.name = name

self.longitude = float(longitude)

self.width = float(width)

if hub != None and hub != False:

self.hub = True

else:

self.hub = False

self.history = history

if self.history != None:

self.year = year

else:

self.year = False

if isSubject != None and isSubject!=False:

self.isSubject = True

else: self.isSubject = False

self.list\_adjacency = []

self.isAdjac = False

self.isBMR = isBMR

def output (self):

result = self.name+' '+str(self.longitude)+' '+str(self.width)+' '

if self.hub:

result+= 'узловая'

if self.history!=None:

result+=' '+self.history

if self.isBMR:

result+= ' BMR '

string\_python\_shit = ''

for x in self.list\_adjacency:

string\_python\_shit = string\_python\_shit+' '+x.name

print (result, ':', string\_python\_shit)

def isEqual(self, obj2):

return (self.name == obj2.name and self.longitude == obj2.longitude and self.width == obj2.width and self.hub == obj2.hub)

def copy(self):

return Station(self.name, self.longitude, self.width, self.hub, self.history, self.year, self.isSubject, self.isBMR)

def compare\_strings(history\_name, station\_name): #нужно, поскольку большинство железнодорожный станций с сайта "Academic" названо иначе, чем в списке от ПЧ

delete\_num = 0

for i in range (len(station\_name)):

i = i-delete\_num

try:

temp = int(station\_name[i])

delete\_num +=1

station\_name = station\_name[:i-1]+station\_name[i+1:]

except:

continue

for i in range (len(history\_name)):

if history\_name[i] == 'ё':

history\_name = history\_name[:i]+'е'+history\_name[i+1:]

temp\_history\_name\_list = history\_name.lower().split(' ')

temp\_station\_name\_list = station\_name.lower().split('.')

history\_name\_list = []

station\_name\_list = []

if len(temp\_history\_name\_list)!=len(temp\_station\_name\_list):

return False

for x in temp\_history\_name\_list:

temp\_history\_list\_kill\_me\_pls = x.split('-')

for y in temp\_history\_list\_kill\_me\_pls:

history\_name\_list.append(y)

for x in temp\_station\_name\_list:

temp\_station\_list\_kill\_me\_pls = x.split('-')

for y in temp\_station\_list\_kill\_me\_pls:

station\_name\_list.append(y)

#if isArsamas:

# print (history\_name\_list, station\_name\_list)

if abs(len(history\_name\_list)-len(station\_name\_list))>1:

return False

for i in range (len(history\_name\_list)):

history\_name\_temp = history\_name\_list[i]

try:

station\_name\_temp = station\_name\_list[i]

if station\_name\_temp != history\_name\_temp[:len(station\_name\_temp)]:

return False

except:

break

return True

if \_\_name\_\_ == '\_\_main\_\_':

wb = openpyxl.load\_workbook('finally\_ready\_station\_file.xlsx')

ws = wb['Sheet']

list\_station = []

for row in ws:

if row[0].value != 'name' and row[0].value != None:

list\_station.append(

Station(row[0].value, row[1].value, row[2].value, row[3].value, row[4].value, row[5].value,

row[6].value, row[7].value))

f = open('result.txt', 'r')

list\_hubs = []

for i in range (1158):

list\_hubs.append(f.readline(i+1))

f.close()

for x in list\_hubs:

for y in list\_station:

if compare\_strings(x, y.name):

y.hub = True

break

wb = openpyxl.Workbook()

ws = wb.active

for x in list\_station:

if x.hub == True:

x.hub = 'узловая'

else:

x.hub = None

ws.append({'A': 'name', 'B': 'долгота', 'C': 'широта', 'D': 'узловая', 'E': 'исторические поселения',

'F': 'год основания', 'G': 'субъект', 'H': 'БМО'})

for x in list\_station:

ws.append({'A': x.name, 'B': x.longitude, 'C': x.width, 'D': x.hub, 'E': x.history, 'F': x.year, 'G': x.isSubject, 'H': x.isBMR})

wb.save('finally\_ready\_station\_file.xlsx'

Код, редактирующий полученные с помощью геокодирования данные, и составляющий базу данных:

import math

import openpyxl

import psycopg2

class Station:

def \_\_init\_\_(self, name, longitude, width, hub, history, year, isSubject, isBMR):

self.unified = False

self.name = name

self.longitude = float(longitude)

self.width = float(width)

if hub != None and hub != False:

self.hub = True

else:

self.hub = False

self.history = history

if self.history != None:

self.year = year

else:

self.year = False

if isSubject != None and isSubject!=False:

self.isSubject = True

else: self.isSubject = False

self.list\_adjacency = []

self.isAdjac = False

self.isBMR = isBMR

def output (self):

result = self.name+' '+str(self.longitude)+' '+str(self.width)+' '

if self.hub:

result+= 'узловая'

if self.history!=None:

result+=' '+self.history

if self.isBMR:

result+= ' BMR '

string\_python\_shit = ''

for x in self.list\_adjacency:

string\_python\_shit = string\_python\_shit+' '+x.name

print (result, ':', string\_python\_shit)

def isEqual(self, obj2):

return (self.name == obj2.name and self.longitude == obj2.longitude and self.width == obj2.width and self.hub == obj2.hub)

def copy(self):

return Station(self.name, self.longitude, self.width, self.hub, self.history, self.year, self.isSubject, self.isBMR)

class Point:

def \_\_init\_\_(self, x, y):

self.x = float(x)

self.y = float(y)

self.isHub = False

self.isDeadEnd = False

self.history = None

self.isInVector = False

def output(self, string\_temp):

print (self.x, self.y, string\_temp)

class LineString:

def \_\_init\_\_(self, init\_point, finish\_point, history):

self.init\_station = init\_point

self.finish\_station = finish\_point

self.list\_stations = []

self.direction = init\_point.x<finish\_point.x

self.history = history

if self.history!=None:

self.init\_station.history = history

self.finish\_station.history = history

self.isInVector = False

def length(self):

return distance(self.init\_station, self.finish\_station)

def output(self, string\_temp):

print(self.init\_station.x, self.init\_station.y, self.finish\_station.x, self.finish\_station.y, string\_temp)

class Vector:

def \_\_init\_\_(self, list\_points):

self.list\_points = list\_points

self.init\_station = find\_closest\_station(list\_points[0])

self.finish\_station = find\_closest\_station(list\_points[-1])

self.list\_result = self.history()

def history(self):

list\_result = []

for x in self.list\_points:

if x.history!=None:

list\_result.append(x.history)

return list\_result

def output(self):

print (self.init\_station.name, '-', self.finish\_station.name, 'VECTOR')

def distance(point1, point2):

return ((point1.x-point2.x)\*\*2+(point1.y-point2.y)\*\*2)\*\*0.5

def count\_for\_point(list\_temp, point\_temp): #считает, сколько раз в данных встречается станция

#исходя из этого, определяем, является ли данная станция узловой с точки зрения теории графов

result = 0

for x in list\_temp:

if distance(x, point\_temp)<0.001:

result +=1

#point\_temp.output(str(result)+' count\_point')

return result

def create\_list\_vectors(): #создает базу данных - список векторов

vector\_list = []

for x in hub\_list:

if not x.isInVector:

temp\_linestring = find\_first\_linestring(x)

temp\_linestring.isInVector = True

temp\_direct = find\_other\_station\_from\_line(x, temp\_linestring)

vector\_list.append(create\_vector(x, temp\_direct))

return vector\_list

def create\_vector(init\_hub, direct\_point): #формирует один вектор

temp\_points\_list = []

temp\_points\_list.append(init\_hub)

temp\_points\_list.append(direct\_point)

temp = direct\_point

init\_hub.output('init\_hub')

direct\_point.output('direct\_point')

init\_hub.isInVector = True

direct\_point.isInVector = True

while not temp.isHub and not temp.isDeadEnd:

temp.output('temp')

temp\_linestring = find\_first\_linestring(temp) #находит для узловой точки какую-то соединенную с ней, после чего циклически идет до следующей узловой точки

temp\_linestring.isInVector = True

temp = find\_other\_station\_from\_line(temp, temp\_linestring)

temp\_points\_list.append(temp)

temp.isInVector = True

vector\_result = Vector(temp\_points\_list)

vector\_result.output()

return vector\_result

def find\_other\_station\_from\_line(point1, line1):

if is\_point\_in\_line(point1, line1) == 1:

return line1.finish\_station

else:

return line1.init\_station

def find\_first\_linestring(point1): #находит соединенную с данной станцию

for x in main\_linestring\_list:

if is\_point\_in\_line(point1, x)>0 and not x.isInVector:

return x

return None

def is\_point\_in\_line(point1, line1): #проверяет, принадлежит ли точка отрезку

if(abs(point1.x-line1.init\_station.x)<0.001 and abs(point1.y - line1.init\_station.y)<0.001):

return 1

elif(abs(point1.x - line1.finish\_station.x)<0.001 and abs(point1.y - line1.finish\_station.y)<0.001):

return 2

else:

return 0

def distance\_mixed(point1, station1):

return distance(point1, Point(station1.longitude, station1.width))

def find\_closest\_station(point1):

min\_r = 5

temp = None

for x in main\_list\_station:

temp\_r = distance\_mixed(point1, x)

if temp\_r<min\_r:

min\_r = temp\_r

temp = x

return temp

def excel\_save(): #для сохранения базы данных с помощью экселя - удобно для дальнейшего представления

wb = openpyxl.Workbook()

ws = wb.active

index = 0

for x in vector\_list:

temp\_l = [x.init\_station.name, x.finish\_station.name, index]

ws.append(temp\_l)

index += 1

for i in range(len(vector\_list)):

ws\_temp = wb.create\_sheet(title=str(i))

for x in vector\_list[i].list\_points:

temp\_l = [x.x, x.y, x.history]

ws\_temp.append(temp\_l)

wb.save('result.xlsx')

if \_\_name\_\_ == '\_\_main\_\_':

excel\_file = openpyxl.load\_workbook('finally\_ready\_station\_file.xlsx')

excel\_file = excel\_file['Sheet']

main\_list\_station = []

for row in excel\_file:

if row[0].value != 'name' and row[0].value != None:

main\_list\_station.append(

Station(row[0].value, row[1].value, row[2].value, row[3].value, row[4].value, row[5].value,

row[6].value, row[7].value))

linestring\_file = openpyxl.load\_workbook('linestring\_data.xlsx')

linestring\_file = linestring\_file['Sheet']

main\_linestring\_list = []

point\_list = []

index = 0

for row in linestring\_file:

if row[0].value != 'x\_init':

temp\_point\_first = Point(row[0].value, row[1].value)

point\_list.append(temp\_point\_first)

temp\_point\_second = Point(row[2].value, row[3].value)

point\_list.append(temp\_point\_second)

main\_linestring\_list.append(LineString(temp\_point\_first, temp\_point\_second, row[4].value))

print(index)

index += 1

hub\_list = []

dead\_end\_list = []

for x in point\_list:

if count\_for\_point(point\_list, x) > 2:

hub\_list.append(x)

x.isHub = True

if count\_for\_point(point\_list, x) == 1:

x.output('DeadEnd')

dead\_end\_list.append(x)

x.isDeadEnd = True

print (len(dead\_end\_list), 'Dead\_end\_list')

vector\_list = create\_list\_vectors()

for x in vector\_list: #находят названия направлений - векторов

x.init\_station = find\_closest\_station(x.list\_points[0])

x.finish\_station = find\_closest\_station(x.list\_points[-1])

with open('./password.txt') as f:

password = f.read()

conn = psycopg2.connect(dbname = 'postgres', user = 'postgres', password = password, host = 'localhost')

cursor = conn.cursor()

cursor.execute('CREATE TABLE IF NOT EXISTS vector\_list (id serial primary key,'

'init\_station text not null, finish\_station text not null)')

for x in vector\_list:

cursor.execute('INSERT INTO vector\_list (init\_station, finish\_station) VALUES (%s, %s)', (x.init\_station.name, x.finish\_station.name))

cursor.execute('CREATE TABLE IF NOT EXISTS linestring\_list (x double precision not null, y double precision not null,'

' history text, id serial, foreign key (id) references vector\_list (id))')

for i in range (len(vector\_list)):

for x in vector\_list[i].list\_points:

cursor.execute('INSERT INTO linestring\_list (x, y, history, id) VALUES (%s, %s, %s, %s)',

(x.x, x.y, x.history, i+1))

conn.commit()

cursor.close()

conn.close()

Код, который занимается алгоритмическим поиском колец:

import openpyxl

import psycopg2

import math

class Station:

def \_\_init\_\_(self, name, longitude, width, hub, history, year, isSubject, isBMR):

self.unified = False

self.name = name

self.longitude = float(longitude)

self.width = float(width)

if hub != None and hub != False:

self.hub = True

else:

self.hub = False

self.history = history

if self.history != None:

self.year = year

else:

self.year = False

if isSubject != None and isSubject!=False:

self.isSubject = True

else: self.isSubject = False

self.list\_adjacency = []

self.isAdjac = False

self.isBMR = isBMR

def output (self):

result = self.name+' '+str(self.longitude)+' '+str(self.width)+' '

if self.hub:

result+= 'узловая'

if self.history!=None:

result+=' '+self.history

if self.isBMR:

result+= ' BMR '

string\_python\_shit = ''

for x in self.list\_adjacency:

string\_python\_shit = string\_python\_shit+' '+x.name

print (result, ':', string\_python\_shit)

def isEqual(self, obj2):

return (self.name == obj2.name and self.longitude == obj2.longitude and self.width == obj2.width and self.hub == obj2.hub)

def copy(self):

return Station(self.name, self.longitude, self.width, self.hub, self.history, self.year, self.isSubject, self.isBMR)

class Point:

def \_\_init\_\_(self, x, y, history):

self.x = float(x)

self.y = float(y)

self.isHub = False

self.isDeadEnd = False

self.history = history

self.isInVector = False

self.isMatched = False

def output(self, string\_temp):

print (self.x, self.y, string\_temp)

class LineString:

def \_\_init\_\_(self, init\_point, finish\_point, history):

self.init\_station = init\_point

self.finish\_station = finish\_point

self.list\_stations = []

self.direction = init\_point.x<finish\_point.x

self.history = history

if self.history!=None:

self.init\_station.history = history

self.finish\_station.history = history

self.isInVector = False

def length(self):

return distance(self.init\_station, self.finish\_station)

def output(self, string\_temp):

print(self.init\_station.x, self.init\_station.y, self.finish\_station.x, self.finish\_station.y, string\_temp)

class Ring\_as\_Line:

def \_\_init\_\_(self, ring):

result = []

for x in ring.list\_of\_vectors:

for y in x.list\_linestrings:

result.append(y)

self.linestring\_list = []

for i in range (1, len(result)):

self.linestring\_list.append(LineString(result[i-1], result[i], None))

class Vector:

def \_\_init\_\_(self, list\_linestrings, init\_station, finish\_station, id):

self.list\_linestrings = list\_linestrings

self.init\_station = init\_station

self.finish\_station = finish\_station

self.id=id

self.isUsed = False

self.isDeadEnd = False

def init\_subject(self):

return find\_subject(self.list\_linestrings[0])

def finish\_subject(self):

return find\_subject(self.list\_linestrings[-1])

def angle(self, direct\_vector, general\_direct):

#general\_direct: 1 - East, 2 - South, 3 - West, 4 - North

if general\_direct == 1:

ideal\_angle = 0

elif general\_direct == 2:

ideal\_angle = math.pi\*1.5

elif general\_direct == 3:

ideal\_angle = math.pi

else:

ideal\_angle = math.pi/2

if direct\_vector:

temp = angle\_vector(self.list\_linestrings[-1], self.list\_linestrings[0])

else:

temp = angle\_vector(self.list\_linestrings[0], self.list\_linestrings[-1])

return abs(temp - ideal\_angle)

def history(self):

list\_history = []

for x in self.list\_linestrings:

if x.history!=None and not x.history in list\_history:

list\_history.append(x.history)

return len(list\_history)

def isDirect(self, station1):

return self.init\_station == station1

def reverse(self):

self.list\_linestrings.reverse()

temp = self.init\_station

self.init\_station = self.finish\_station

self.finish\_station = temp

def isNormalVector(self):

return self.init\_station != self.finish\_station

def output(self):

print (self.init\_station+' - '+self.finish\_station, 'vector')

def length(self):

return distance(self.list\_linestrings[0], self.list\_linestrings[-1])

def x(self):

return self.list\_linestrings[-1].x-self.list\_linestrings[0].x

def y(self):

return self.list\_linestrings[-1].y-self.list\_linestrings[0].y

class Subject:

def \_\_init\_\_(self, north\_border, south\_border, west\_border, east\_border, name, saturation, weight, id):

self.north\_border = north\_border

self.south\_border = south\_border

self.west\_border = west\_border

self.east\_border = east\_border

self.centre = Point((west\_border+east\_border)/2, (north\_border+south\_border)/2, None)

self.name = name

self.saturation = saturation

self.weight = weight

self.id = id

def isinSubject(self, point1):

return point1.x>=self.west\_border and point1.x<=self.east\_border and point1.y>self.south\_border and point1.y<self.north\_border

class Ring:

def \_\_init\_\_(self, list\_of\_vectors, capital\_name, subject\_name, capital\_point):

self.list\_of\_vectors = list\_of\_vectors

self.capital\_name = capital\_name

self.subject\_name = subject\_name.split(' ')[0]

self.capital\_point = capital\_point

def history(self):

result = 0

for x in self.list\_of\_vectors:

result+=x.history()

return result

def distance\_to\_edge(self):

r\_max = 0

for x in self.list\_of\_vectors:

for y in x.list\_linestrings:

temp = distance(y, self.capital\_point)

if temp>r\_max:

r\_max = temp

return r\_max

def distance\_longitude(self): #максимальная разница в долготе на маршруте

r\_max = 0

for x in self.list\_of\_vectors:

for y in x.list\_linestrings:

temp = abs(y.x-self.capital\_point.x)

if temp>r\_max:

r\_max =temp

return r\_max

def distance\_width(self): #максимальная разница в широте на маршруте

r\_max = 0

for x in self.list\_of\_vectors:

for y in x.list\_linestrings:

temp = abs(y.y-self.capital\_point.y)

if temp>r\_max:

r\_max = temp

return r\_max

def isNormal(self): #проверяет, соответствует ли кольцо критериям адекватности

temp\_distance\_edge = self.distance\_to\_edge()

temp\_history = self.history()

print (self.capital\_name, temp\_distance\_edge, temp\_history, 'ring arguments')

ratio = self.distance\_longitude()/self.distance\_width()

return temp\_distance\_edge>1.5 and temp\_history>3 and ratio<3 and ratio>1/3

def time\_to\_finish(length, capital\_point, that\_point):

return distance(that\_point, capital\_point)<2 and length>8

def find\_subject(point1): #проверяет, к какому субъекту относится точка

list\_result = []

for x in subject\_list:

if x.isinSubject(point1):

list\_result.append(x)

if len(list\_result)==0:

return find\_closest\_subject(point1, subject\_list)

return find\_closest\_subject(point1, list\_result)

def angle\_vector(station1, station2):

width1 = station1.y

width2 = station2.y

longitude1 = station1.x

longitude2 = station2.x

try:

tan = (width1-width2)/(longitude1-longitude2)

except:

tan = (width1-width2)/(longitude1-longitude2+1)

isTanPos = tan>=0

isCosPos = longitude1>=longitude2

if (isTanPos and isCosPos): #I четверть

return math.pi/2-math.atan(tan)

elif isTanPos and not isCosPos: #III четверть

return math.pi\*1.5-math.atan(tan)

elif not isTanPos and isCosPos: #IV четверть

return math.pi/2-math.atan(tan)

else: #II четверть

return math.pi\*1.5-math.atan(tan)

def select\_list(cursor1): #формирует список векторов

list\_vector = []

index = 0

for row in cursor1:

temp\_id = row[0]

init\_station = row[1]

finish\_station = row[2]

cursor\_temp = conn.cursor()

cursor\_temp.execute('SELECT \* FROM linestring\_list WHERE id = ' + str(temp\_id) + '')

list\_linestring = []

for row\_linestring in cursor\_temp:

list\_linestring.append(Point(row\_linestring[0], row\_linestring[1], row\_linestring[2]))

list\_vector.append(Vector(list\_linestring, init\_station, finish\_station, temp\_id))

#print (init\_station, finish\_station)

index+=1

if index == 1:

list\_vector[0].isDeadEnd = True

return list\_vector

def distance(point1, point2):

return ((point1.x - point2.x) \*\* 2 + (point1.y - point2.y) \*\* 2) \*\* 0.5

def sort\_for\_capitals():

list\_result = []

for x in main\_list\_station:

if x.isSubject:

list\_result.append(x)

return list\_result

def find\_closest\_subject(point1, subject\_list):

r\_min = 5

temp = None

for x in subject\_list:

temp\_r = distance(x.centre, point1)

if temp\_r<r\_min:

temp = x

r\_min = temp\_r

return temp

def history\_sort():

list\_result = []

for x in main\_list\_station:

if x.history:

list\_result.append(x)

return list\_result

def create\_history\_list():

list\_result = []

cursor\_temp = conn.cursor()

cursor\_temp.execute('SELECT history FROM linestring\_list')

for row\_temp in cursor\_temp:

list\_result.append(row\_temp[0])

return list\_result

def correct\_history\_subject\_list(temp\_list):

list\_result = []

for x in temp\_list:

if x in history\_list:

list\_result.append(x)

return list\_result

def list\_important\_points(temp\_history\_list): #составляет список интересных исторически точек

list\_result = []

for x in temp\_history\_list:

temp\_vector = find\_vector\_for\_hist(x)

for y in temp\_vector:

list\_result.append(y)

return list\_result

def find\_vector\_for\_hist(history\_name): #проверяет, интересен ли вектор с точки зрения исторических поселений

temp\_cursor = conn.cursor()

id\_list = []

temp\_cursor.execute('SELECT \* FROM linestring\_list where history like \''+history\_name+'%\'')

for temp\_row in temp\_cursor:

point\_temp = Point(temp\_row[0], temp\_row[1], temp\_row[2])

id\_list.append(temp\_row[3])

if isSimilar(id\_list):

temp\_cursor.execute('SELECT \* FROM linestring\_list where id = '+str(id\_list[0]))

count = 0

index = 0

point\_first = None

point\_last = None

for temp\_row in temp\_cursor:

count+=1

temp\_cursor.execute('SELECT \* FROM linestring\_list where id = ' + str(id\_list[0]))

for temp\_row in temp\_cursor:

if index == 0:

point\_first = Point(temp\_row[0], temp\_row[1], temp\_row[2])

if index == count-1:

point\_last = Point(temp\_row[0], temp\_row[1], temp\_row[2])

index+=1

return [point\_first, point\_last]

else:

return [point\_temp]

def isSimilar(temp\_list):

temp = temp\_list[0]

for x in temp\_list:

if x!=temp:

return False

return True

def find\_best\_actual\_point(point\_list, capital\_point): #находит точку, направление к которой минимальное (см. компас)

angle\_min = 7

temp\_point = None

for x in point\_list:

temp\_angle = angle\_vector(x, capital\_point) #здесь норм

if temp\_angle<angle\_min and not x.isMatched:

temp\_point = x

angle\_min = temp\_angle

temp\_point.isMatched = True

return temp\_point

def reverse\_vector(vector1, init\_station\_name):

if init\_station\_name != vector1.init\_station:

vector1.reverse()

return vector1

else:

return vector1

def isNonPriorVector(vector1, vector\_temp\_vector):

if vector1.length()>1.8 and vector1.history()<3:

return True

if len(vector\_temp\_vector) == 1 and vector1.finish\_station == vector\_temp\_vector[0].init\_station:

return True

for i in range(1, len(vector\_temp\_vector)):

if vector1.finish\_station == vector\_temp\_vector[i].init\_station:

return True

return False

def length\_ring(temp\_ring):

result = 0

for x in temp\_ring:

result+=x.length()

return result

def isDeadEnd(name): #проверяет, тупиковая ли станция

cursor\_temp = conn.cursor()

cursor\_temp.execute(

'SELECT \* FROM vector\_list WHERE init\_station = \'' + name + '\' or finish\_station = \'' + name + '\'')

index = 0

for row in cursor\_temp:

index+=1

return index == 1

def find\_best\_vector(init\_station\_name, init\_point, finish\_point, vector\_temp\_list, temp\_ring, isBMR):

#ищет вектор с наиболее близким к идеальному вектору направлением

best\_angle = 7

cursor\_temp = conn.cursor()

cursor\_temp.execute('SELECT \* FROM vector\_list WHERE init\_station = \''+init\_station\_name+'\' or finish\_station = \''+init\_station\_name+'\'')

temp\_list = select\_list(cursor\_temp)

temp\_vector = None

print (len(temp\_list), 'length')

print (init\_station\_name)

for x in temp\_list:

x = reverse\_vector(x, init\_station\_name)

#здесь перебрать тупики

choice\_vector = []

for x in temp\_list:

if not isDeadEnd(x.finish\_station):

choice\_vector.append(x)

elif x.history()>0 and not isNonPriorVector(x, temp\_ring):

print('history\_dead\_end')

return x

non\_priority\_vector\_list = []

try:

capital\_point = temp\_ring[0].list\_linestrings[0]

except:

capital\_point = init\_point

for x in choice\_vector:

temp\_angle = double\_angle(x, Vector([init\_point, finish\_point], None, None, 0))

#print(temp\_angle, 'angle')

#x.output()

if temp\_angle<best\_angle and x.isNormalVector() and isActualVector(x, vector\_temp\_list):

try:

if isNonPriorVector(x, temp\_ring) and not time\_to\_finish(length\_ring(temp\_ring), capital\_point, init\_point):

non\_priority\_vector\_list.append(x)

else:

if not (finish\_ring(x, temp\_ring[0].init\_station, isBMR) and length\_ring(temp\_ring)<5):

best\_angle = temp\_angle

temp\_vector = x

except:

try:

if not (finish\_ring(x, temp\_ring[0].init\_station, isBMR) and length\_ring(temp\_ring)<5):

best\_angle = temp\_angle

temp\_vector = x

except:

best\_angle = temp\_angle

temp\_vector = x

if temp\_vector == None:

try:

return non\_priority\_vector\_list[0]

except:

return None

#print (length\_ring(temp\_ring), 'length\_ring')

return temp\_vector

def isEqualPoint(point1, point2):

return distance(point1, point2)<0.1

def find\_closest\_station(point1):

r\_distance = 1

temp\_station = None

for x in main\_list\_station:

temp = distance(point1, Point(x.longitude, x.width, None))

if temp<r\_distance:

temp\_station = x

r\_distance = temp

#работает

return temp\_station

def find\_BMR(point1):

return find\_closest\_station(point1).isBMR #проверяет, является ли станция частью Большого Московского Кольца

def finish\_ring(vector1, capital\_name, isBMR): #проверяет, замкнут ли маршрут

if isBMR and find\_BMR(vector1.list\_linestrings[-1]):

return True

return vector1.finish\_station == capital\_name

def from\_hist\_to\_hist(init\_station\_name, init\_point, finish\_point, temp\_ring, isBMR): #цикл построения кольцевого маршрута от одной исторической точки до другой

vector\_list = []

temp\_vector = find\_best\_vector(init\_station\_name, init\_point, finish\_point, vector\_list, temp\_ring, isBMR)

temp\_ring.append(temp\_vector)

vector\_list.append(temp\_vector)

temp\_vector.output()

while not isEqualPoint(temp\_vector.list\_linestrings[-1], finish\_point):

if length\_ring(temp\_ring)>30:

return None

temp\_vector = find\_best\_vector(temp\_vector.finish\_station,

Point(temp\_vector.list\_linestrings[-1].x, temp\_vector.list\_linestrings[-1].y, None), finish\_point, vector\_list, temp\_ring, isBMR)

if temp\_vector == None:

return None

vector\_list.append(temp\_vector)

temp\_ring.append(temp\_vector)

temp\_vector.output()

if isRing(temp\_ring):

print ('isRing')

break

return temp\_ring

def isActualList(list\_point):

for x in list\_point:

if x.isMatched:

return True

return False

def isRing(ring\_list): #проверяет, является ли маршрут замкнутым

try:

if find\_BMR(ring\_list[0].list\_linestrings[0]) and find\_BMR(ring\_list[-1].list\_linestrings[-1]) and length\_ring(ring\_list)>1:

return True

return ring\_list[0].init\_station == ring\_list[-1].finish\_station

except IndexError:

return False

def find\_for\_BMR(capital\_name, capital\_point): #ищет направление для идеального вектора - противоположное центру Москвы

#нужно, чтобы маршрут не протекал долго по Большому Московскому Кольцу

cursor\_temp = conn.cursor()

cursor\_temp.execute('SELECT \* FROM vector\_list WHERE init\_station = \'' + capital\_name + '\' or finish\_station = \'' + capital\_name + '\'')

temp\_list = select\_list(cursor\_temp)

init\_vector = Vector([capital\_point, Moscow\_point], capital\_name, 'Москва', 1)

angle\_max = 0

temp\_point = None

for x in temp\_list:

x = reverse\_vector(x, capital\_name)

for x in temp\_list:

temp\_angle = double\_angle(init\_vector, x)

if angle\_max<temp\_angle and x.length()>0.1 and distance(capital\_point, x.list\_linestrings[0])<0.1:

angle\_max = temp\_angle

temp\_point = x.list\_linestrings[-1]

print (temp\_angle, x.finish\_station, 'temp\_angle')

temp\_point.output('temp\_point')

return temp\_point

def create\_ring(capital\_point, temp\_important\_list, capital\_name, subject\_name, isBMR):

temp\_point = find\_best\_actual\_point(temp\_important\_list, capital\_point)

ring\_result = []

temp\_name = capital\_name

temp\_init\_point = capital\_point

if isBMR: #исключение для Москвы как начальной точки маршрута

ring\_result = from\_hist\_to\_hist(capital\_name, capital\_point, find\_for\_BMR(capital\_name, capital\_point), ring\_result, isBMR)

temp\_init\_point = ring\_result[-1].list\_linestrings[-1]

temp\_name = ring\_result[-1].finish\_station

while not isRing(ring\_result):

try:

if time\_to\_finish(length\_ring(ring\_result), capital\_point, ring\_result[-1].list\_linestrings[-1]):

temp\_point = capital\_point

except:

print ('first\_vector')

ring\_result = from\_hist\_to\_hist(temp\_name, temp\_init\_point, temp\_point, ring\_result, isBMR)

if ring\_result == None:

return None

if isRing(ring\_result):

break

temp\_point.isMatched = True

temp\_point = find\_best\_actual\_point(temp\_important\_list, capital\_point)

temp\_name = ring\_result[-1].finish\_station

temp\_init\_point = ring\_result[-1].list\_linestrings[-1]

print (temp\_name)

return Ring(ring\_result, capital\_name, subject\_name, capital\_point)

def isActualVector(vector1, vector\_list):

for x in vector\_list:

if vector1.init\_station == x.init\_station and vector1.finish\_station == x.finish\_station:

return False

return True

def double\_angle(vector1, vector2): #начинаются с одной point

#x1\*x2+y1\*y2=|vector1|\*|vector2|\*cos(angle)

dot\_products = vector1.x()\*vector2.x()+vector1.y()\*vector2.y()

try:

cos\_angle = dot\_products/(vector1.length()\*vector2.length())

except:

cos\_angle = 1

return math.acos(round(cos\_angle, 4)) #считает угол с помощью скалярного произведения векторов

if \_\_name\_\_ == '\_\_main\_\_':

Moscow\_point = Point(37.6, 55.7, 'Москва')

used\_vectors = []

excel\_file = openpyxl.load\_workbook('./finally\_ready\_station\_file.xlsx')

excel\_file = excel\_file['Sheet']

main\_list\_station = []

with open('./password.txt') as f:

password = f.read()

with open ('./host.txt') as f:

host = f.read()

print (host)

for row in excel\_file:

if row[0].value != 'name' and row[0].value != None:

main\_list\_station.append(Station(row[0].value, row[1].value, row[2].value, row[3].value, row[4].value, row[5].value,

row[6].value, row[7].value))

conn = psycopg2.connect(dbname='postgres', user='postgres', password=password, host= host)

cursor = conn.cursor()

capital\_list = sort\_for\_capitals()

subject\_list = []

history\_list = create\_history\_list()

cursor.execute('SELECT \* FROM subject\_rf')

for row in cursor:

subject\_list.append(Subject(row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7]))

ring\_list = []

for x in capital\_list: #перебирает столицы субъектов - для каждой пытается строить кольцевой маршрут

temp\_subject = find\_subject(Point(x.longitude, x.width, None))

temp\_hist\_list = []

cursor.execute('SELECT history FROM subject\_history where id = '+ str(temp\_subject.id))

for row in cursor:

temp\_hist\_list.append(row[0])

temp\_hist\_list = correct\_history\_subject\_list(temp\_hist\_list)

print (temp\_hist\_list)

important\_list = list\_important\_points(temp\_hist\_list)

if len(temp\_hist\_list) > 2:

test\_ring = create\_ring(Point(x.longitude, x.width, None), important\_list, x.name, temp\_subject.name, x.isBMR)

if test\_ring!=None:

if test\_ring.isNormal(): #проверяет, соответствует ли кольцо критериям адекватности

ring\_list.append(test\_ring)

print ('Х') #если ты понимаешь, о чем я

index = 0

wb = openpyxl.Workbook()

for x in ring\_list:

temp\_table\_name = x.subject\_name + '\_' + str(index)+'\_vector\_name'

ws\_temp = wb.create\_sheet(x.subject\_name+'\_'+str(index)+'\_кольцо')

for y in x.list\_of\_vectors:

ws\_temp.append([y.init\_station, y.finish\_station])

cursor.execute(

'CREATE TABLE IF NOT EXISTS ' + temp\_table\_name + '(vector\_id serial not null, init\_station text, finish\_station text);')

for y in x.list\_of\_vectors:

cursor.execute('INSERT INTO ' + temp\_table\_name + ' VALUES (%s, %s, %s)',

(y.id, y.init\_station, y.finish\_station))

temp\_table\_name = x.subject\_name+'\_'+str(index)+'\_visual'

cursor.execute('CREATE TABLE IF NOT EXISTS '+temp\_table\_name+'(linestring geometry not null)')

temp\_visual = Ring\_as\_Line(x)

for x in temp\_visual.linestring\_list:

cursor.execute('INSERT INTO '+temp\_table\_name+' VALUES (st\_makeline(st\_makepoint('+

str(x.init\_station.x)+', '+str(x.init\_station.y)+

'), st\_makepoint('+str(x.finish\_station.x)+', '+str(x.finish\_station.y)+')))')

index+=1

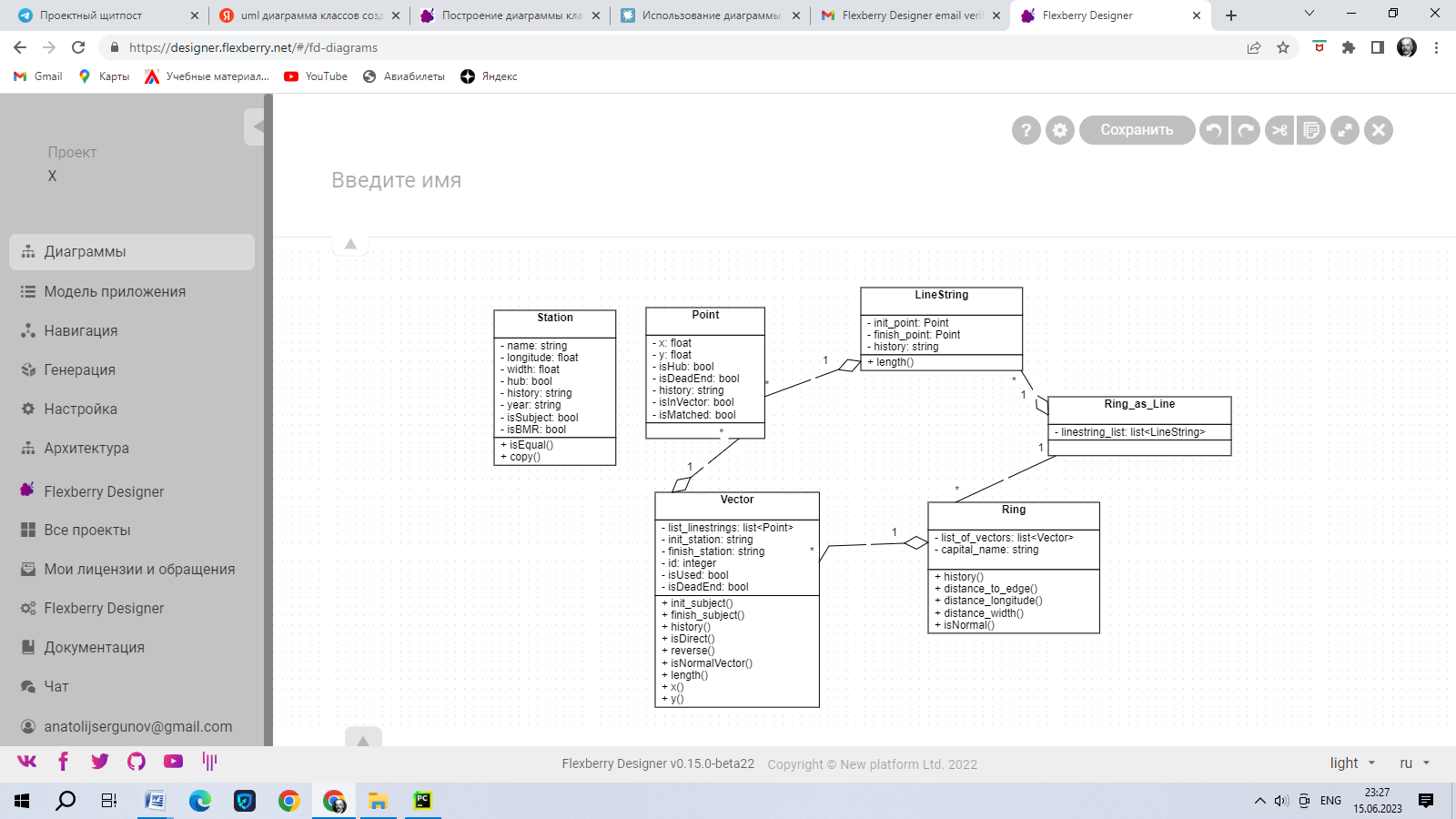
wb.save ('Excel\_rings.xlsx')

cursor.close()

conn.commit()

conn.close()

**Uml-диаграмма алгоритма**

****