Отчет по лабораторной работе №2

по курсу «Алгоритмика»

Выполнил:

Григорьев Илья

517 группа

Москва, 2022

Постановка задачи

Имеется связный неориентированный граф G=(V,E), где V – множество 196 городов России, E – множество ребер, образованных всеми парами городов. Для каждого города известны географические координаты и для каждого ребра известна длина в километрах. Эти данные представлены в виде таблицы.

Битоническим маршрутом называется замкнутый путь из самого западного города в самый восточный и обратно, в котором первая часть маршрута проходится строго слева направо (с запада на восток), а вторая часть – строго справа налево (с востока на запад). При этом каждый город кроме первого посещается строго по одному разу. Нужно разработать $O(n^2)$ алгоритм определения кратчайшего битонического маршрута через все города «от Калининграда до Магадана и обратно».

Описание данных

Данные представлены в виде таблицы: для каждого города указаны его географические координаты и расстояние до каждого другого города.

Описание метода решения

Для решения данной задачи было решено использовать динамическое программирование. Вначале отсортируем все города с запада на восток. Далее будем двигаться по отсортированному массиву. На каждом шаге есть два варианта: либо этот город посещается на пути с запада на восток, либо этот город посещается на обратном пути. Вычисляем стоимости каждого из этих двух вариантов рекурсивно: как сумму расстояний на текущем шаге и результат рекурсивного вызова функции на подзадаче. Далее берем наименьшее полученное расстояние, сохраняем его в массив (главный принцип динамического программирования), так же сохраняем в массив трассировку городов, чтобы восстановить маршрут, и возвращаем результат. Разумеется, что при каждом вызове функции проверяется условие выхода из рекурсии (когда дошли до Магадана) и проверяется наличие уже посчитанной подзадачи в массиве.

Разработанный алгоритм работает со сложностью $O(n^2)$. Сортировка занимает $O(n\,logn)$ времени и уходит в о-малое по отношению к общей сложности алгоритма. Далее вызов функции из начальной точки и все

последующие рекурсивные вызовы занимают $O(n^2)$ времени, так как двумя параметрами функции «current» и «target» производится двойной обход массива городов. С точки зрения сложности алгоритмов это эквивалентно двойному циклу по массиву городов. Трассировка маршрута не добавляет сложности, так как все необходимые действия выполняются в тех же вызовах рекурсивной функции. Итого получаем общую сложность метода равную $O(n^2)$.

Описание программой реализации

Программа была реализована в Jupyter Notebook на языке программирования Python3. Все ячейки кода подписаны – что именно там выполняется. Для запуска программы необходимо иметь установленный Python3, Jupyter Notebook и все зависимости. Далее надо последовательно выполнить все ячейки кода. В программе реализовано: чтение исходных данных из Excel файла, препроцессинг, основной алгоритм решения задачи, вывод времени работы, вывод длины кратчайшего маршрута, вывод последовательности городов в порядке посещения и визуализация результата на карте.

Эксперименты

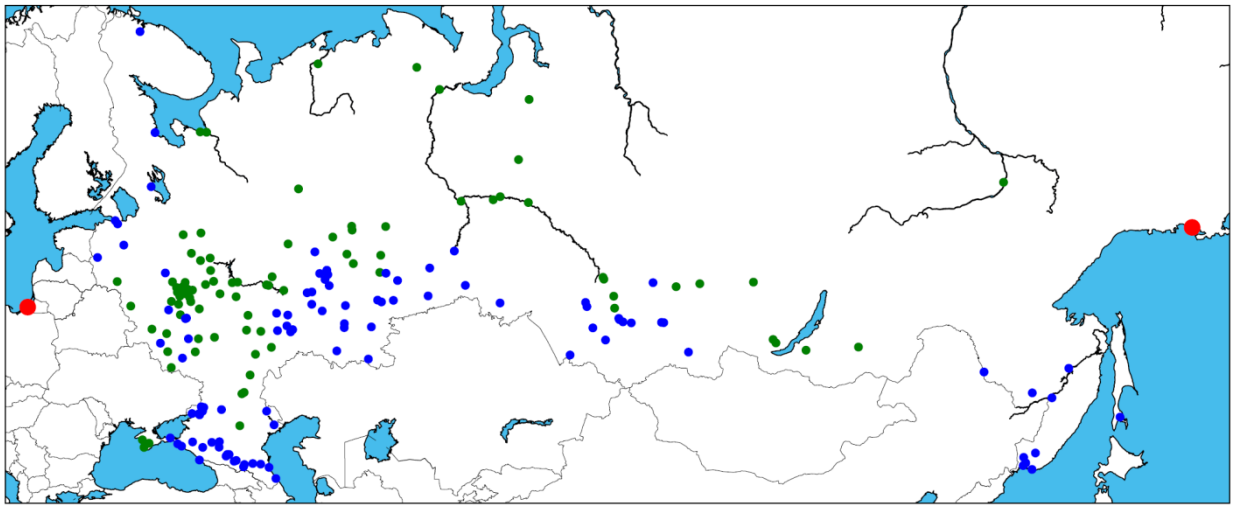
Полученный результат: 95055км. Время работы алгоритма: 90мс.

Города в порядке посещения с запада на восток: ['Калининград', 'Великие Луки', 'Смоленск', 'Евпатория', 'Севастополь', 'Симферополь', 'Брянск', 'Орёл', 'Курск', 'Белгород', 'Обнинск', 'Клин', 'Зеленоград', 'Одинцово', 'Красногорск', 'Серпухов', 'Химки', 'Подольск', 'Тула', 'Москва', 'Мытищи', 'Королёв', 'Пушкино', 'Люберцы', 'Череповец', 'Балашиха', 'Щёлково', 'Железнодорожный', 'Жуковский', 'Сергиев Посад', 'Раменское', 'Ногинск', 'Электросталь', 'Воскресенск', 'Коломна', 'Рыбинск', 'Орехово-Зуево', 'Воронеж', 'Липецк', 'Рязань', 'Северодвинск', 'Ярославль', 'Вологда', 'Владимир', 'Архангельск', 'Кострома', 'Иваново', 'Ковров', 'Тамбов', 'Муром', 'Дзержинск', 'Арзамас', 'Нижний Новгород', 'Элиста', 'Волгоград', 'Волжский', 'Пенза', 'Саранск', 'Камышин', 'Саратов', 'Кузнецк', 'Чебоксары', 'Новочебоксарск', 'Балаково', 'Йошкар-Ола', 'Казань', 'Киров', 'Сыктывкар', 'Нарьян-Мар', 'Кудымкар', 'Пермь', 'Соликамск', 'Березники', 'Кунгур', 'Первоуральск', 'Нижний Тагил', 'Серов', 'Воркута', 'Салехард', 'Ханты-Мансийск', 'Нефтеюганск', 'Сургут', 'Ноябрьск', 'Нижневартовск', 'Новый

Уренгой', 'Северск', 'Томск', 'Кемерово', 'Ленинск-Кузнецкий', 'Красноярск', 'Канск', 'Братск', 'Ангарск', 'Иркутск', 'Улан-Удэ', 'Чита', 'Якутск', 'Магадан'].

Города в порядке посещения с востока на запад: ['Магадан', 'Южно-Сахалинск', 'Комсомольск-на-Амуре', 'Хабаровск', 'Арсеньев', 'Биробиджан', 'Находка', 'Артём', 'Уссурийск', 'Владивосток', 'Благовещенск', 'Кызыл', 'Минусинск', 'Абакан', 'Ачинск', 'Междуреченск', 'Новокузнецк', 'Прокопьевск', 'Киселёвск', 'Бийск', 'Барнаул', 'Бердск', 'Новосибирск', 'Рубцовск', 'Омск', 'Ишим', 'Тобольск', 'Тюмень', 'Курган', 'Каменск-Уральский', 'Челябинск', 'Екатеринбург', 'Миасс', 'Златоуст', 'Магнитогорск', 'Орск', 'Уфа', 'Стерлитамак', 'Салават', 'Оренбург', 'Нефтекамск', 'Чайковский', 'Воткинск', 'Сарапул', 'Октябрьский', 'Ижевск', 'Глазов', 'Набережные Челны', 'Альметьевск', 'Нижнекамск', 'Самара', 'Новокуйбышевск', 'Димитровград', 'Тольятти', 'Сызрань', 'Ульяновск', 'Дербент', 'Астрахань', 'Махачкала', 'Харабали', 'Хасавюрт', 'Грозный', 'Назрань', 'Владикавказ', 'Нарткала', 'Нальчик', 'Пятигорск', 'Ессентуки', 'Кисловодск', 'Волгодонск', 'Ставрополь', 'Невинномысск', 'Армавир', 'Шахты', 'Майкоп', 'Новочеркасск', 'Новошахтинск', 'Батайск', 'Сочи', 'Ростов-на-Дону', 'Краснодар', 'Таганрог', 'Елец', 'Новомосковск', 'Узловая', 'Старый Оскол', 'Новороссийск', 'Анапа', 'Керчь', 'Калуга', 'Тверь', 'Железногорск', 'Беломорск', 'Петрозаводск', 'Мурманск', 'Великий Новгород', 'Колпино', 'Санкт-Петербург', 'Псков', 'Калининград'].

Визуализация результата на карте (красные точки – Калининград и Магадан, зеленые точки – города на маршруте с запада на восток, синие точки – города на маршруте с востока на запад):



Выводы

Для решения битонической евклидовой задачи коммивояжёра было применено динамическое программирование: оптимизация рекурсивного алгоритма с помощью запоминания и повторного использования результатов уже посчитанных подзадач. Используя данный подход удалось разработать эффективный и интерпретируемый алгоритм решения поставленной задачи, работающий со сложностью $O(n^2)$. Так же была реализована визуализация результата на карте с помощью библиотек matplotlib и mpl_toolkits.