МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики — процессов управления

Проект по предмету
«Теория конечных графов и её приложения»
Анализ карты города Уфа (OpenStreetMap)

Выполнили студенты 17.Б11-пу:
Ахметов Аскар Ильгизович
Сокол Милена Денисовна
Щелкина Оксана Темуриевна

Санкт-Петербург
2020

Оглавление

Постановка задачи

Точная постановка задачи, данная преподавателем.

Перед началом работы получить граф выбранного города (Уфы). Координаты объектов взять по местоположению реальных объектов города. Расстояние между реальным положением объекта и ближайшим узлов графа пренебречь. За расстояние между вершинами графа принять кратчайшее расстояние между вершинами графа,

1. Оценка удобства размещения объектов инфраструктуры города.

   На карте случайным образом выбраны M объектов (больниц) и N узлов (жилых домов). Величины M и N равны 10 и 100 соответственно.

   1.1. Для каждого узла (дома):
      1.1.1. Определить ближайший от узла объект (путь «туда»), ближайший к объекту узел (путь «обратно»), объект расстояние до которого и обратно минимально («туда и обратно»).
      1.1.2. Определить объекты, расположенные не далее, чем в X км для каждого из трех вариантов «туда», «обратно», «туда и обратно».
   1.2. Определить, какой из объектов расположен так, что расстояние между ним и самым дальним домом минимально («туда», «обратно», «туда и обратно»).
   1.3. Определить объект, для которого сумма кратчайших расстояний от него до всех домов минимальна.
   1.4. Определить объект, для которого построенное дерево кратчайших путей имеет минимальный вес.

2. Планирование новых объект

   На карте случайным образом выбраны N узлов (домов) и один из объектов инфраструктуры.

   2.1. Построить дерево кратчайших путей от объекта до выбранных узлов.
   2.2. Вычислить общую длину дерева, а также сумму кратчайших расстояний от объекта до всех заданных узлов.
   2.3. Разбить выбранные узлы на кластеры, используя метод полной связи (complete-linkage clustering). Построить дендрограмму разбиения узлов.
      2.3.1. Пусть узлы разбиты на k кластеров.
      2.3.2. Найти расположение центра масс (центроида) для каждого кластера;
      2.3.3. Построить дерево кратчайших путей от объекта до центроидов.
      2.3.4. Для каждого кластера построить дерево кратчайших путей от центроида до всех вершин кластера.Найти длину построенного дерева и сумму кратчайших расстояний от объекта до всех заданных узлов. .
   2.4. Сравнить найденные в п.1 и 3 величины для k=2, 3, 5

Выделение подзадач.

Для упрощения работы объемная задача была разбита на следующие более мелкие задачи:

1. Подготовительный этап - подключение библиотек; создание необходимых структур; выделение 10 больниц и 100 домов из списка всех вершин; написание функций визуализации, алгоритма Дейкстры; сохранение списка, матрицы смежности и матрицы кратчайших путей.
2. Реализация первого задания - работа с матрицами кратчайших путей; визуализация и сохранение результатов; построение деревьев.
3. Реализация второго задания - работа с кластеризацией; визуализация и сохранение результатов; построение деревьев.
4. Тестирование - запуск алгоритмов на тестовых данных, предоставленных преподавателями; сравнение результатов работы реализованных самостоятельно алгоритмов с алгоритмами, реализованными в крупных библиотеках (например, networkx).

Использованные технологии.

В качестве языка программирования был выбран Python 3.7, поскольку все участники команды были уже знакомы с данным языком. Помимо этого для Python создано множество библиотек, которые упрощают работы и с графами, и конкретно с картами OSM.

В качестве среды разработки предпочтение отдавалось Google Colab, поскольку в нем возможна параллельная работа нескольких человек, но также использовался и Anaconda Jupyter Notebook.

Ниже перечислены библиотеки, которые были задействованы в проекте:

1. osmnx - работа с OSM-картами, визуализация графов;
2. networkx - работа с графами, создание матриц и списков смежности;
3. matplotlib - построение графиков кластеров и центроид в кластерах;
4. numpy - удобное представление данных;
5. pandas - сохранение матрицы смежности, работа с тестовыми данными;
6. random - выбор рандомных 10 больниц и 100 домов;
7. csv - сохранение результатов для предоставления преподавателем, работа с тестовыми данными:
8. heapq - функции pop и push для работы со структурой Heap в реализации алгоритма Дейкстры;
9. scipy

Разделение обязанностей.

Во время обсуждения работы над проектом, команда решила, что удобнее будет не дробить обязанности, соответственно как такового разделения не было.

Написание кода осуществлялось в общем ноутбуке на Google Colab, поэтому каждый в любой момент мог дополнить код сокомандника, указать на ошибки и т.д.

Подготовительный этап

В качестве города была выбрана Уфа (население: 1128787 (на 2020 год), площадь: 707,93 км^2. Для получения графа города использовалась библиотека osmnx.

Рисунок 1. Визуализация графа Уфы.

Далее были созданы матрица смежности и список смежности для всего графа. Количество вершин в графе составило ~25000.

Следующим осуществлялся выбор необходимого количества вершин графа, с которыми будет проводиться дальнейшая работа.

В качестве объектов возьмем больницы (hospitals). Выберем их всех существующих зданий 10 больниц и 100 многоквартирных домов (apartments).

В качества алгоритма для поиска кратчайших путей из заданной функции мы взяли алгоритм Дейкстры (поскольку в исходном графе отсутствуют ребра отрицательного веса).

Было сделано несколько реализаций, но в итоге самой быстрой оказалась последняя – использующая кучу (heap). В качестве опоры мы использовали реализацию данного алгоритма в библиотеке networkx, с которым сравнивали итоговые результаты, чтобы убедиться в правильности.

Поскольку реализация нашего алгоритма Дейкстры основывалась на реализации одноимённого алгоритма из networkx (nx.dijkstra_path), то производительность и точность у алгоритмов практически совпадает. Данный алгоритм реализован с помощью бинарной кучи.

В отличии от алгоритма из networkx реализованный нами алгоритм возвращает не только лист кратчайших путей от source до target, а также и длину данного пути (для

этого в networkx есть отдельная функция nx.dijkstra_path_length, возвращающая только длину пути).

Например, для нахождения N + M путей из выбранных вершин до всех остальных, алгоритм затрачивает ~40 секунд, что является очень хорошим результатом.

Поскольку алгоритм Дейкстры строит пути по всему графу, они включают вершины, которых нет в списках домов или больниц.

Мы решили удалять лишние вершины так, чтобы расстояние все равно сохранялось.

Пусть вершины A и D принадлежат выбранным N + M вершинам, а вершины B и C - нет. Тогда сокращение путей сработает следующим образом:

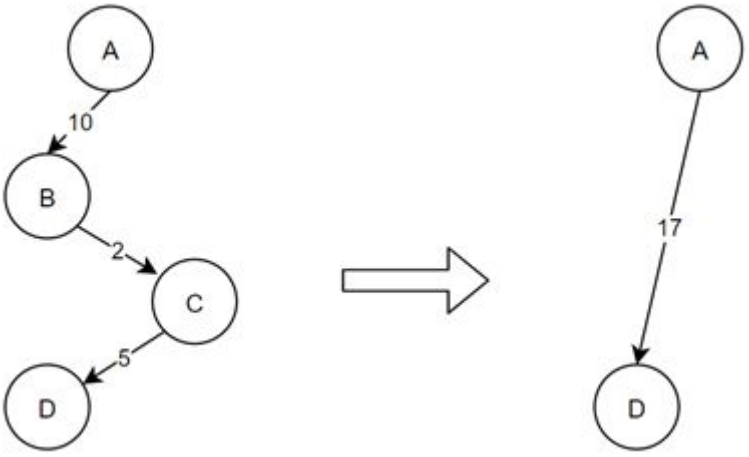

Рисунок 2. Пример сокращения путей в графе.

На основе результатов алгоритма Дейкстры (после сокращения) построим матрицу кратчайших путей для выбранных N + M вершин и сохраним ее в csv.

```
array([[1.00e+10, 1.16e+02, 1.21e+02, ..., 6.40e+01, 2.20e+01, 6.20e+01],
       [1.16e+02, 1.00e+10, 7.60e+01, ..., 1.19e+02, 9.50e+01, 1.21e+02],
       [1.42e+02, 3.60e+01, 1.00e+10, ..., 1.55e+02, 1.21e+02, 1.47e+02],
       ...,
       [6.80e+01, 1.11e+02, 1.47e+02, ..., 1.00e+10, 7.00e+01, 1.10e+02],
       [2.40e+01, 1.03e+02, 9.90e+01, ..., 7.40e+01, 1.00e+10, 4.00e+01],
       [6.40e+01, 1.18e+02, 1.14e+02, ..., 1.14e+02, 4.00e+01, 1.00e+10]])
```

Рисунок 3. Матрица кратчайших путей.

С помощью полученной матрицы кратчайших путей несложно получить деревья кратчайших путей из любой из N + M вершин. Для анализа результатов были использованы встроенные функции из библиотек osmnx, networkx и matplotlib, визуализирующие граф и пути на нем. Для примера возьмем самую первую вершину из в списка всех вершин и рассмотрим полученное для нее дерево кратчайших путей.

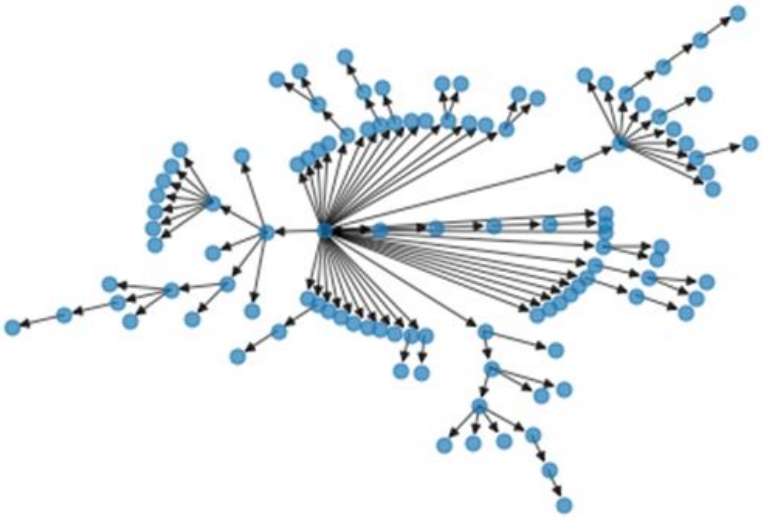

Рисунок 4. Дерево кратчайших путей для вершины 1.

Рисунок 5. Дерево кратчайших путей на графе Уфы для вершины 1.

Результаты.

Задание 1.

Результатом выполнения задания 1.1.a является матрица [3x100], каждая строка которой соответствует одному из выбранных домов, а каждая строка содержит в себе ближайшую от дома больницу (путь «туда»), ближайший к больнице дом (путь «обратно»), больницу, расстояние до которой и обратно минимально («туда и обратно»).

В качестве примера мы вывели на исходном графе путь из некоторой вершины до больницы, расстояние до которой и обратно минимально.

Рисунок 6. Визуализация произвольного пути между двумя вершинами, расстояние «туда и обратно» между которыми минимально.

После выполнения задания 1.1.b мы получили лист листов, содержащий в себе для каждого дома вершины-больницы, расположенные не далее, чем в X км для каждого из трех вариантов «туда», «обратно», «туда и обратно».

Рисунок 7. Визуализация произвольного пути между двумя вершинами, между которыми расстояние < 50.

При выполнении следующего пункта в результате мы получили координаты вершины нужной больницы. Отметили ее на карте и построили из нее дерево кратчайших путей.

Рисунок 8. Визуализация объекта на графе, расположенного так, что расстояние между ним и самым дальним домом минимально.

Рисунок 9. Визуализация дерева кратчайших путей из ранее найденного объекта

Задание 2.

1. Для полученного дерева кратчайших путей были вычислены длина построенного дерева и сумма кратчайших расстояний от объекта (больницы) до всех заданных узлов (домов).

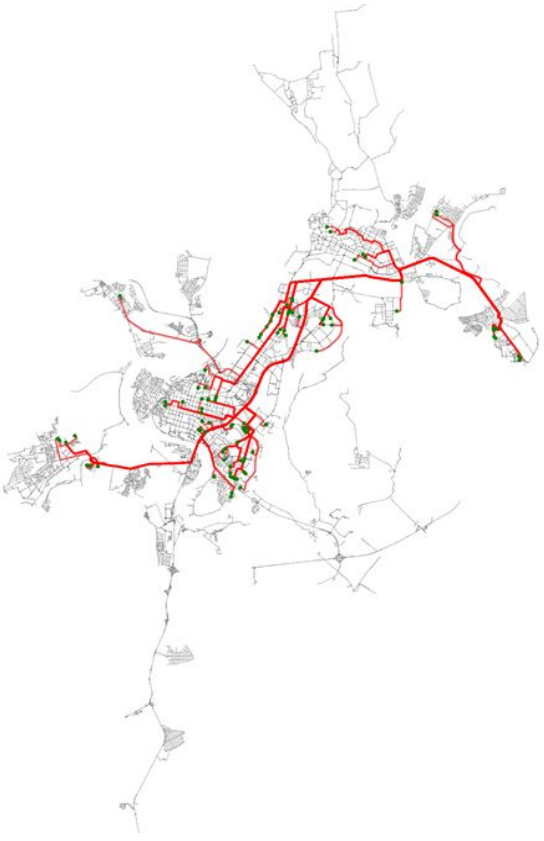

Рисунок 10. Дерево кратчайших путей от объекта до выбранных узлов(домов).

```
hosp_index
```

```
3700686070
```

```
weights_of_hosp
```

```
7630
```

```
sum_shortest_paths
```

```
11001.0
```

Рисунок 11. Результаты вычислений общей длины дерева, а также суммы кратчайших расстояний от объекта до всех заданных узлов.

2. Далее была проведена кластеризация методом полной связи. Для начала дома были разбиты на кластеры без указания количества необходимых кластеров. В результате мы получили следующую дендрограмму разбиения:

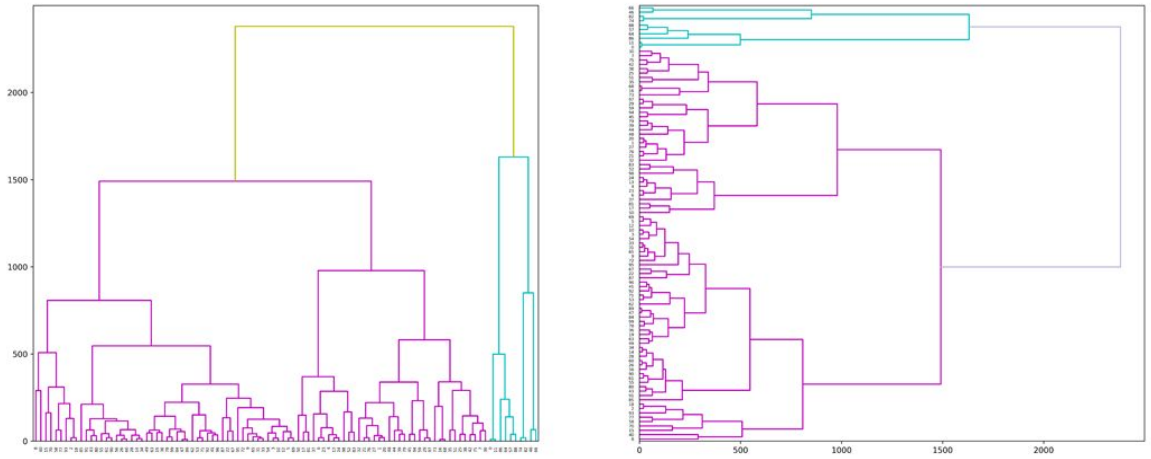

Рисунок 12. Дендрограмма разбиения узлов.

3. Далее уже рассматривалась кластеризация с параметром k. который отвечал за будущее количество кластеров.

В полученных кластерах следом выделялись центроиды.

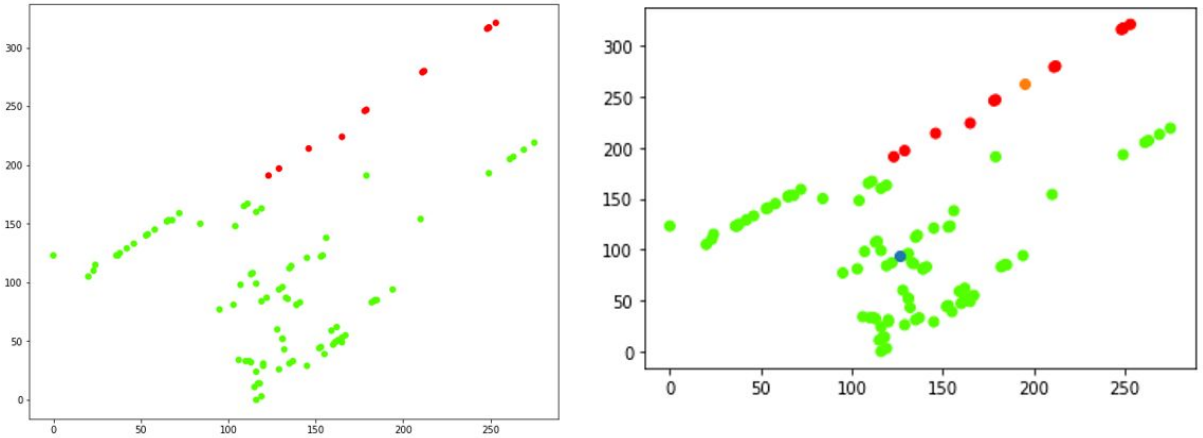

Рисунок 13. Разбиение точек на два кластера и выделение центроид.

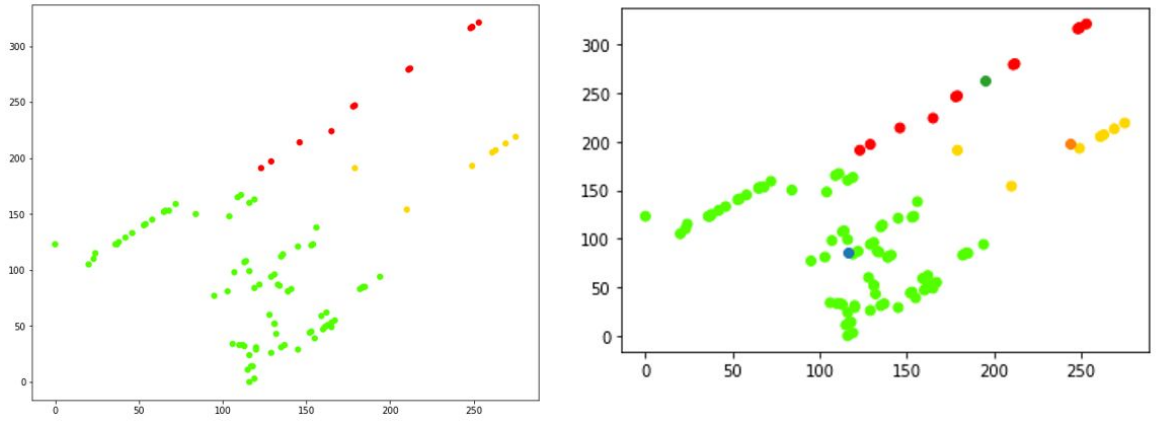

Рисунок 14. Разбиение точек на три кластера и выделение центроид.

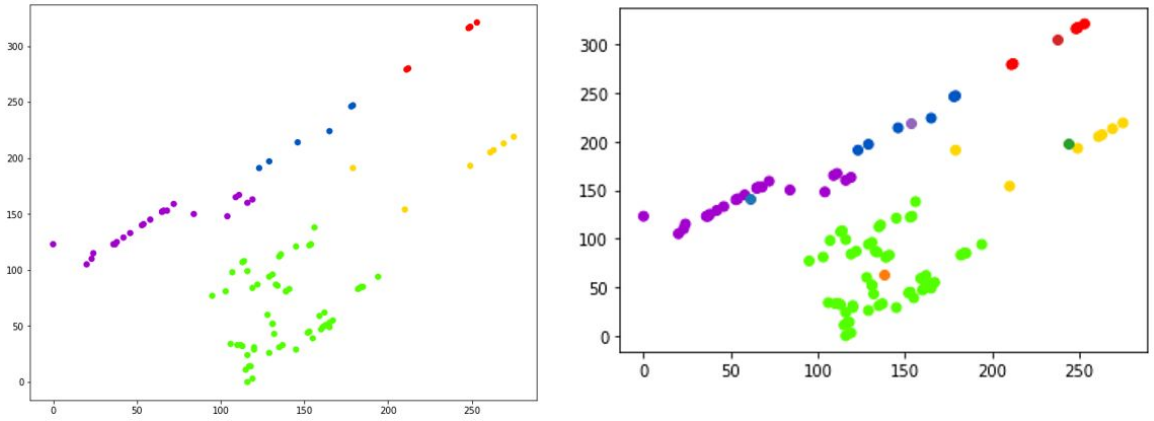

Рисунок 15. Разбиение точек на пять кластеров и выделение центроид.

` Следующим шагом было построение дерева кратчайших путей от больницы, выбранной в пункте 2.1 до полученных центроид. Результаты также представлены для трех вариантов значения параметра k (k=2,3,5).

Рисунок 16. Визуализация дерева кратчайших путей от объекта до центроидов (k=2).

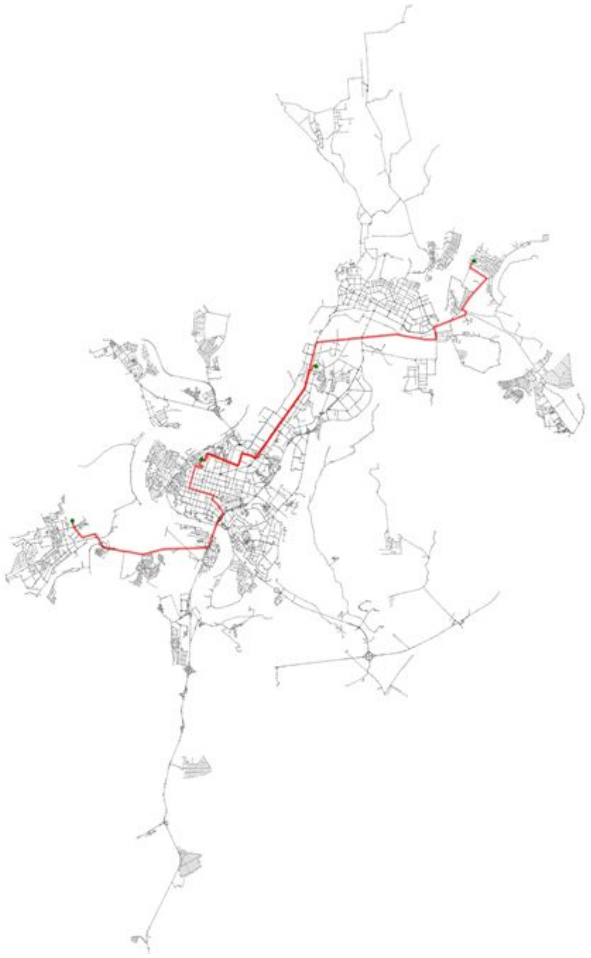

Рисунок 17. Визуализация дерева кратчайших путей от объекта до центроидов (k=3).

Рисунок 18. Визуализация дерева кратчайших путей от объекта до центроидов (k=5).

Далее для каждого k были построены деревья кратчайших путей от центроид кластеров до остальных вершин, принадлежавших ему.

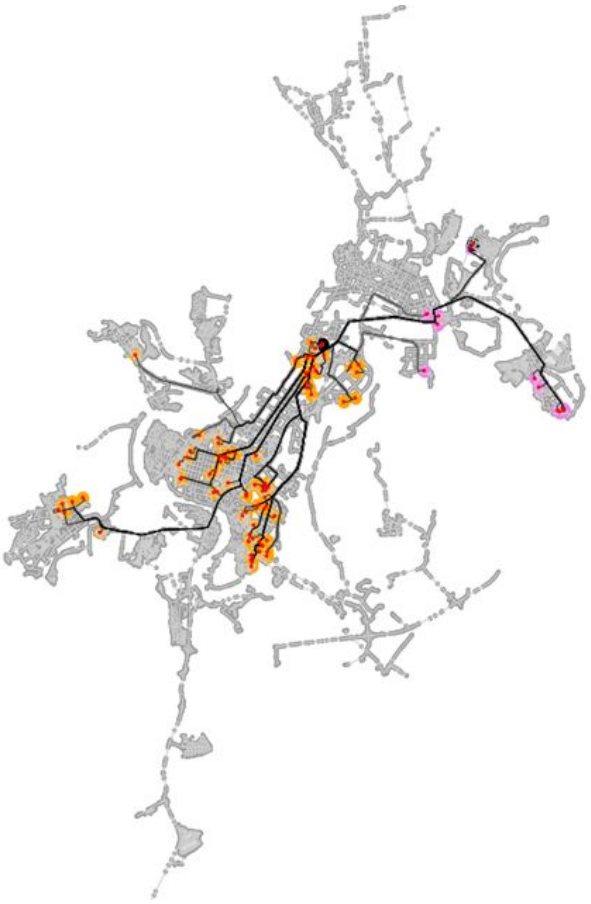

Рисунок 19. Визуализация дерева кратчайших путей от центроидов до всех вершин кластера (k = 2).

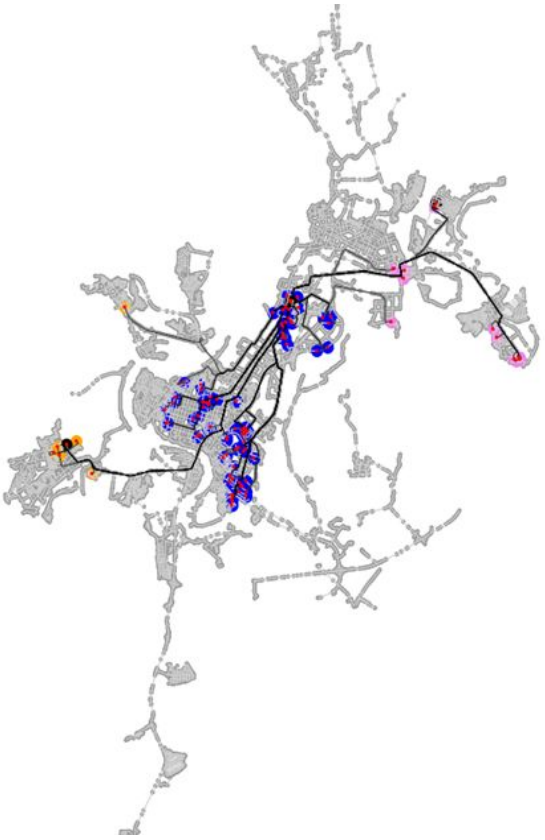

Рисунок 20. Визуализация дерева кратчайших путей от центроидов до всех вершин кластера (k = 3).

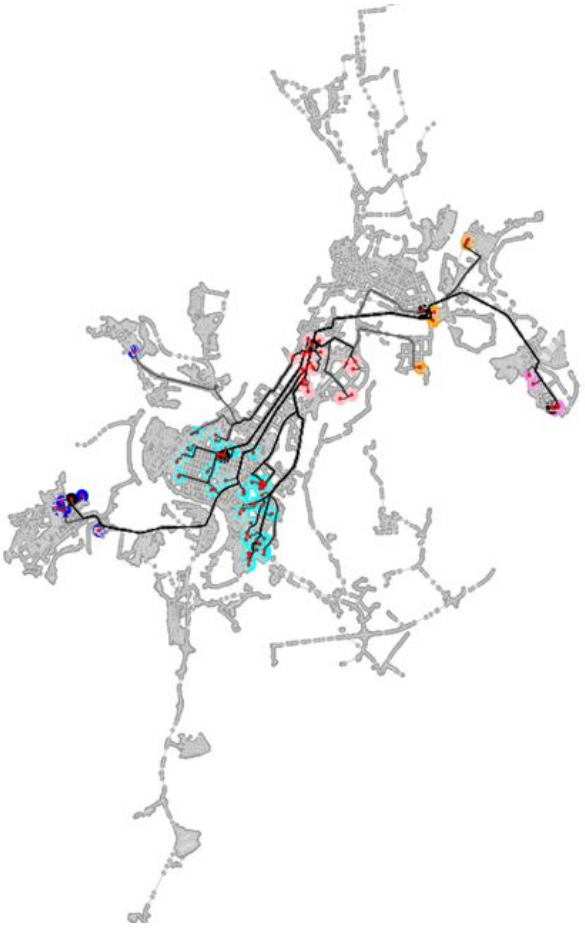

Рисунок 21. Визуализация дерева кратчайших путей от центроидов до всех вершин кластера (k = 5).

4. Для всех полученных деревьев кратчайших путей были вычислены длина построенного дерева и сумма кратчайших расстояний от объекта (больницы) до всех заданных узлов (домов).

```
weights_of_hosp, sum_shortest_paths
```

```
(7661, 12923)
```

```
weight_centroids_tree, sum_shortest_paths_centroids
```

```
(34927, 100543)
```

```
weight_centroids_tree, sum_shortest_paths_centroids
```

```
(64076, 176487)
```

Рисунок 22. Результаты задания 2.3.d для k = 2, 3, 5

Эти результаты были сравнены с результатами, полученными в пункте 2.1. На их основе был проведен анализ и сделаны следующие выводы.

Тестирование.

Для тестирования выполненного проекта (в частности алгоритма Дейкстры) было предложено запустить код на выданных преподавателями тестовых данных, а именно матриц смежности в csv форматах (для вершин 3 на 3, 5 на 5, 8 на 8, 400 на 400 и 10000 на 10000).

Были выполнены и загружены матрицы кратчайших путей для матрицы 3 на 3, 5 на 5, 8 на 8 и 400 на 400. Для матрицы 10000 на 10000 построение деревьев кратчайших путей оказалось невозможным из-за того, что не хватало либо мощности, либо если мощности хватало, то результат занимал очень много памяти и программа падала. Максимальным результатом были деревья кратчайших путей из 1000 точек до 10000. В csv был загружен результат 1 на 10000.

Матрица 3 на 3:

Алгоритм Дейкстры был запущен из трех вершин.
Время работы примерно 0.00018 с.

```
array([[  0,  98, 224],
       [ 98,   0, 322],
       [224, 322,   0]])
```

Рисунок 23. Запуск на матрице 3 на 3.

Матрица 5 на 5:

Алгоритм Дейкстры был запущен из пяти вершин.
Время работы примерно 0.00133 с.

```
array([[0, 3, 1, 2, 3],
       [3, 0, 2, 3, 4],
       [1, 2, 0, 1, 2],
       [2, 3, 1, 0, 3],
       [3, 4, 2, 3, 0]])
```

Рисунок 24. Запуск на матрице 5 на 5.

Матрица 8 на 8:

Алгоритм Дейкстры был запущен из восьми вершин.
Время работы примерно 0.00125 с.

```
array([[  0,  98, 224, 456, 210, 312, 269, 331],
       [ 98,   0, 322, 554, 308, 410, 367, 429],
       [224, 322,   0, 232, 434, 536, 493, 555],
       [456, 554, 232,   0, 666, 768, 725, 787],
       [210, 308, 434, 666,   0, 102,  59, 121],
       [312, 410, 536, 768, 102,   0, 161, 223],
       [269, 367, 493, 725,  59, 161,   0,  62],
       [331, 429, 555, 787, 121, 223,  62,   0]])
```

Рисунок 25. Запуск на матрице 8 на 8.

Матрица 400 на 400:

Алгоритм Дейкстры был запущен из четырехсот вершин.
Время работы примерно 0.98046 с.

```
array([[  0,  98, 224, 456, 210, 312, 269, 331],
       [ 98,   0, 322, 554, 308, 410, 367, 429],
       [224, 322,   0, 232, 434, 536, 493, 555],
       [456, 554, 232,   0, 666, 768, 725, 787],
       [210, 308, 434, 666,   0, 102,  59, 121],
       [312, 410, 536, 768, 102,   0, 161, 223],
       [269, 367, 493, 725,  59, 161,   0,  62],
       [331, 429, 555, 787, 121, 223,  62,   0]])
```

Рисунок 26. Запуск на матрице 400 на 400.

Матрица 10000 на 10000:

Алгоритм Дейкстры был запущен из одной вершины.
Время работы примерно 0.18947 с.

```
array([    0,   276,   343, ..., 13746, 13827, 13846])
```

Рисунок 27. Запуск на матрице 10000 на 10000.

Анализ полученных результатов.

Рассматривая результаты первого пункта, можно сделать вывод, что для каждого дома из выбранных точек в принципе относительно недалеко находится одна из больниц (до которой можно добраться и от которой возможно доехать до дома). Это означает, что в целом объекты инфраструктуры расположены вполне приемлемо.

Используя кластеризацию можно получить точки, через которые оптимальнее всего можно будет добраться от больниц до домов. Расстояния от домов до подобных точек будут минимальны, так что при проектировании дорог будет выгодно уменьшать кратчайший путь от больниц до этих точек. Мы нашли упомянутые точки, как центроиды кластеров домов.

Опираясь на теоретические знания и интуитивные соображения, проходя через центроиды мы должны были получить деревья меньшего веса, чем пункте 2.1, так как мы идем через вершины, которые должны быть максимально приближены к точкам из кластеров.

К сожалению, в нашем проекте мы таких результатов не наблюдаем. Точно осознать в чем проблема нам не удалось, возможно, дело в неправильном выборе центроид или каких-то других неучтенных факторах.

Перечень использованных источников.

1. OpenStreetMap wiki
2. Документация библиотеки osmnx

3. Документация библиотеки networkx
4. Кормен «Алгоритмы. Построение и анализ»
5. Статья с реализацией Дейкстры (бинарная куча через priority_queue)
6. Примеры красивой визуализации в osmnx
7. Документация библиотеки scipy.claster