ПРОЕКТ ПО ПРЕДМЕТУ «ТЕОРИЯ КОНЕЧНЫХ ГРАФОВ И ЕЕ ПРИЛОЖЕНИЯ»

АНАЛИЗ КАРТЫ ГОРОДА УФА (OPENSTREETMAP)

Выполнили студенты 331 группы:
Ахметов Аскар
Сокол Милена
Щелкина Оксана

ИСПОЛЬЗОВАННЫЕ ТЕХНОЛОГИИ

Язык программирования Python 3.7

Среда разработки: Anaconda Jupyter Notebook, Google Colab

Библиотеки:

- osmnx
- networkx
- matplotlib
- numpy
- pandas
- random
- csv
- heapq
- scipy

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.

```
place = {'city' : 'Ufa',
         'country' : 'Russia'}
G = ox.graph_from_place(place, network_type='drive', simplify=False)

fig, ax = ox.plot_graph(G, dpi=500, save=True, close=True, file_format='png',
                        filename='ufa', fig_height = 30, fig_width = 30,
                        node_color='#000000', node_size=2, edge_color='#999999',
                        edge_linewidth=0.7)
```

ПОДГОТОВИТЕЛЬНЫЙ ЭТАП

Сохраним узлы полученного дерева:

```
nodes_df = pd.DataFrame(G.nodes())
nodes_df.to_csv('nodes.csv', index=False)
```

Создадим и сохраним список смежности и матрицу смежности всего графа

```
adj_list = nx.generate_adjlist(G, delimiter=' ')
with open('adjacency_list.csv', "w", newline='') as csv_file:
    writer = csv.writer(csv_file, delimiter=' ')
    for line in adj_list:
        writer.writerow(line)
```

```
G_pd = nx.to_pandas_adjacency(G)
G_pd.to_csv('matrix_adjacency_pandas.csv')
```

ПОДГОТОВИТЕЛЬНЫЙ ЭТАП

В качестве объектов возьмем больницы (hospitals).

Выберем их всех существующих зданий 10 больниц и 100 многоквартирных домов (apartments).

```
# вытаскиваем все больницы и дома
buildings = ox.footprints.footprints_from_place(place, footprint_type='building', retain_invalid=False, which_result=1)

hospitals = []
apartments = []
build = buildings['building'].to_dict()

for key,value in build.items():
    if value == 'hospital':
        hospitals.append(key)
    elif value == 'apartments':
        apartments.append(key)
```

ПОДГОТОВИТЕЛЬНЫЙ ЭТАП

```
# для каждой больницы и дома находим соответсвующие им (ближайшие) ноды на графе
a = buildings.to_dict()
hospitals_dict = {}
apartments_dict = {}

for i in hospitals:
    bounds = a['geometry'][i].bounds
    nearest_node = ox.get_nearest_node(G, ((bounds[1]+bounds[3])/2, (bounds[0]+bounds[2])/2))
    hospitals_dict[i] = nearest_node

for i in apartments:
    bounds = a['geometry'][i].bounds
    nearest_node = ox.get_nearest_node(G, ((bounds[1]+bounds[3])/2, (bounds[0]+bounds[2])/2))
    apartments_dict[i] = nearest_node
```

```
M = 10
N = 100

hospitals_dict = dict(random.sample(list(hospitals_dict.items()), M))
apartments_dict = dict(random.sample(list(apartments_dict.items()), N))
```

{89665061: 7512139535, 95408906: 1237275154, ...101312101: 1393833498} - больницы
{89400291: 2145263454, 101472176: 1186962359.....104988570: 1210433183} - дома

ПОИСК КРАТЧАЙШИХ ПУТЕЙ

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

```
from heapq import heappush, heappop

def dijkstra_heap (G, source):

    G_succ = G._succ

    push = heappush
    pop = heappop
    weight = lambda d: min(attr.get(weight, 1) for attr in d.values())

    seen = {}  # минимальное расстояние до ключа-точки
    dist = {}
    path = {}
    # fringe is heapq with 2-tuples (distance,node)
    fringe = []

    seen[source] = 0
    push(fringe, (0, source, None))

    while fringe:

        (d, v, pred) = pop(fringe)

        if v in dist:
            continue  # already searched this node.

        dist[v] = d

        if pred != None:

          if len(path[pred]) > 0:
            path[v] = (path[pred]).copy()
            path[v].append(pred)
          else:
            path[v] = [pred]
        else:
          path[source] = []

        for u, e in G_succ[v].items():
            cost = weight(e) #e - ребро (мультиграф же,значит их несколько, выбираем наименьший вес)

            if cost is None: # видимо если ребра нет, возможно стоит заменить на бесконечное значение
                continue

            vu_dist = dist[v] + cost

            if u not in seen or vu_dist < seen[u]: #если мы еще не искали путь до u или новое расстояние меньше найденного
                seen[u] = vu_dist
                push(fringe, (vu_dist, u, v))
    return path, dist
```

ПРОИЗВОДИТЕЛЬНОСТЬ

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 кучи.

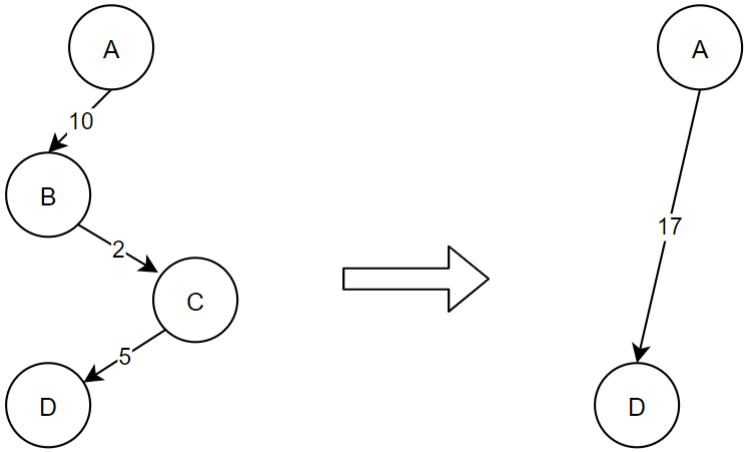
В отличии от алгоритма из networkx реализованный нами алгоритм возвращает не только лист кратчайших путей от source до target, а так же и длину данного пути (для этого в networkx есть отдельная функция nx.dijkstra_path_length, возвращающая только длину пути).

Например, для нахождения 110 путей из выбранных вершин до всех остальных, алгоритм затрачивает ~40 секунд, что является очень хорошим результатом.

СОКРАЩЕНИЕ ПУТЕЙ

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.




```
def reduce_paths (paths, new_nodes):

  small_paths = paths.copy()

  for key, node_paths in paths.items():

    if key not in new_nodes:
      del small_paths[key]
      continue

    keys = list(node_paths.keys())
    small_node_paths = node_paths.copy()

    for child in keys:
      if child not in new_nodes:
        del small_node_paths[child]

    keys = list(small_node_paths.keys())

    for child in keys:
      temp = small_node_paths[child].copy()
      for el in small_node_paths[child]:
        if el not in new_nodes:
          temp.remove(el)
      small_node_paths[child] = temp

    small_paths[key] = small_node_paths
  return small_paths
```

МАТРИЦА КРАТЧАЙШИХ ПУТЕЙ

На основе результатов алгоритма Дейкстры посмотрим матрицу кратчайших путей для N + M вершин и сохраним ее в csv.

```
short_path_matrix = np.zeros((len(new_nodes), len(new_nodes)))

for i in range(N+M):
  dist = dists[new_nodes[i]]
  for j in range(N+M):
    if new_nodes[j] in small_paths[new_nodes[i]]:
      short_path_matrix[i][j] = dist[new_nodes[j]]
    else:
      short_path_matrix[i][j] = 0
```

```
with open('matrix_dijkstra.csv', "w", newline='') as csv_file:
    writer = csv.writer(csv_file, delimiter=',')
    for line in short_path_matrix:
        writer.writerow(line)
```

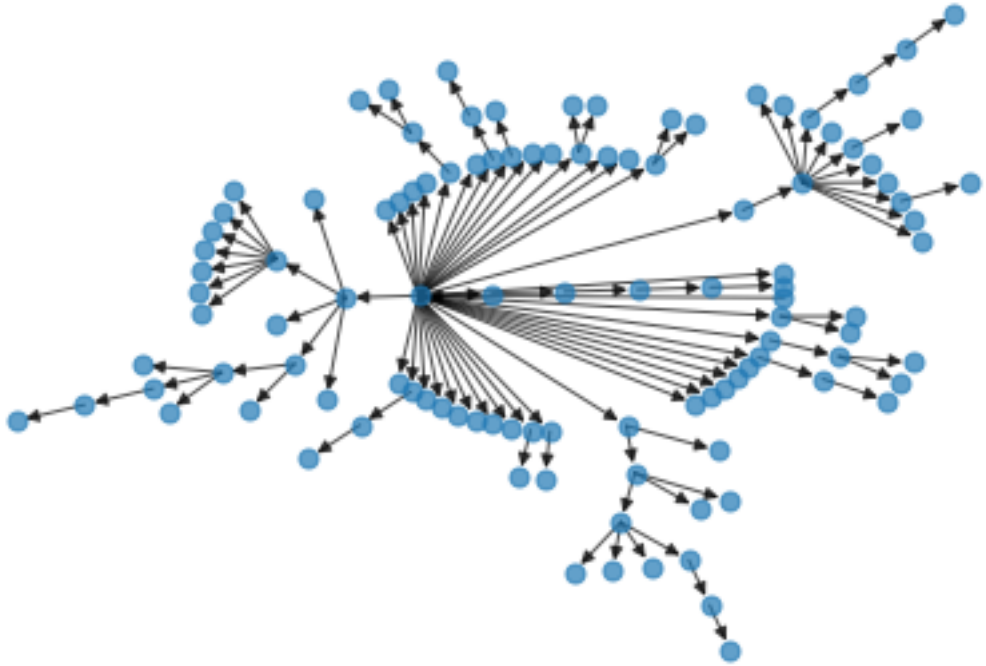
```
array([[1.00e+10, 1.16e+02, 1.21e+02, ..., 6.40e+01, 2.20e+01, 6.20e+01],
       [1.16e+02, 1.00e+10, 7.60e+01, ..., 1.19e+02, 9.50e+01, 1.21e+02],
       [1.42e+02, 3.60e+01, 1.00e+10, ..., 1.55e+02, 1.21e+02, 1.47e+02],
       ...,
       [6.80e+01, 1.11e+02, 1.47e+02, ..., 1.00e+10, 7.00e+01, 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]])
```

ДЕРЕВО КРАТЧАЙШИХ ПУТЕЙ ИЗ ПРОИЗВОЛЬНОЙ ВЕРШИНЫ

Для начала выполним визуализацию дерева кратчайших путей для одной вершины – первой в списке выбранных нами вершин.

Будем использовать функции из библиотеки networkx.

```
small_adjancecy = adjancecy_m_dict[new_nodes[0]]

new_G = nx.from_numpy_matrix(small_adjancecy, create_using = nx.DiGraph)

from networkx.drawing.nx_pydot import graphviz_layout
pos = graphviz_layout(new_G, prog='twopi')
nx.draw(new_G, pos, node_size = 50, alpha = 0.7)
```



ВИЗУАЛИЗАЦИЯ КРАТЧАЙШЕГО ПУТИ ДЛЯ ПРОИЗВОЛЬНОЙ ВЕРШИНЫ

```
route = nx.shortest_path(G,
                         892683760,
                         498826321,
                         weight='length')

ox.plot_graph_route(G, route, dpi=500, node_color='#000000', node_size=2, edge_color='#999999', edge_linewidth=0.2)
```

Изобразим кратчайших путь между двумя произвольными вершинами из списка на исходном графе города.

Конец и начало пути обозначены красными жирными точками, сам путь – красной линией.

ВИЗУАЛИЗАЦИЯ ДЕРЕВА КРАТЧАЙШЕГО ПУТИ НА ГРАФЕ

routes представляет собой лист из листов, каждый из которых является путём на исходном графе.

Поскольку нам необходимо дерево, в routes лежит 109 листов-путей.

```
some_tree = small_paths[new_nodes[0]]
routes = []
for el in new_nodes:
  if el == new_nodes[0]:
    continue
  route = paths[new_nodes[0]][el]
  routes.append(route )
```

```
len(routes)
```

```
109
```

```
ox.plot_graph_routes(G, routes, save=True, close=True, file_format='png',
                     filename='short_path_tree', dpi=500, fig_height = 20,
                     fig_width = 20, dpi=500, node_color='#000000',
                     orig_dest_node_size=0.5, edge_color='#999999',
                     edge_linewidth=0.7)
```

ЗАДАНИЕ 1

Пункт 1.а

Для каждого дома определить ближайший от узла объект (путь “туда”), ближайший к объекту узел (путь “обратно”), объект, расстояние до которого и обратно минимально (“туда и обратно”).

nearest_hosp_list представляет собой лист из листов, каждый из которых соответствует одному из выбранных ста домов и хранить в себе все необходимые вычисленные значения.

Левый столбец – «туда», столбец по середине – «обратно», правый столбец – «туда и обратно».

```
nearest_hosp_list = [[0 for i in range(3)] for j in range(N)]
temp = short_path_matrix.copy()


for i in range(N):
  node = apartments_values[i]
  index = np.where(new_nodes == node)
  # туда
  nearest_hosp_list[i][0] = new_nodes[np.reshape(temp[index][:], 110).argmin()]
  # обратно
  nearest_hosp_list[i][1] = new_nodes[np.reshape(temp[:][index], 110).argmin()]
  #туда и обратно
  nearest_hosp_list[i][2] = new_nodes[(np.reshape(temp[index][:], 110) +
                                       np.reshape(temp[:][index], 110)).argmin()]
```

ЗАДАНИЕ 1

Пункт 1.а

Для каждого дома определить ближайший от узла объект (путь “туда”), ближайший к объекту узел (путь “обратно”), объект, расстояние до которого и обратно минимально (“туда и обратно”).

ЗАДАНИЕ 1

Пункт 1.b

Для каждого дома определить объекты, расположенные не далее, чем в X км для каждого из трех вариантов “туда”, “обратно”, “туда и обратно”.

```
[[[1393833498, 2661495390, 3246184010, 2334187973],
  [1393833498, 2661495390, 3246184010, 2334187973],
  [1393833498]],
 [[1393833498,
   2661495390,
   7512139535,
   1237275208,
   3693060753,
   826163319,
   3246184010,
   2334187973,
   1237275154],
```

```python
def find_in_radius(short_path_matrix, radius, apartments_values, new_nodes):

    permissible_hosps = [[[] for i in range(3)] for j in range(len(apartments_values))]

    for node in apartments_values:
        index = np.where(new_nodes == node)[0][0]
        for j in range(len(new_nodes)):
            if short_path_matrix[index][j] <= radius:
                permissible_hosps[index][0].append(new_nodes[j])
            if short_path_matrix[j][index] <= radius:
                permissible_hosps[index][1].append(new_nodes[j])
            if short_path_matrix[index][j] + short_path_matrix[j][index] <= radius:
                permissible_hosps[index][2].append(new_nodes[j])

    return permissible_hosps
```

```python
radius = 100
hosps_in_radius = find_in_radius(data, radius, apartments_values, new_nodes)
```

ЗАДАНИЕ 1

Пункт 1.b

Для каждого дома определить объекты, расположенные не далее, чем в X км для каждого из трех вариантов “туда”, “обратно”, “туда и обратно”.

```
routes = []
for hosp in hosps_in_radius[0][2]:
  route = paths[apartments_values[0]][hosp]
  if len(route)>0:
    routes.append(route)

ox.plot_graph_routes(G, routes, save=True, close=True, file_format='png',
                     filename='1b_example_route', dpi=500, fig_height = 30,
                     fig_width = 30, node_color='#000000', node_size = 1,
                     orig_dest_node_size=100, edge_color='#999999',
                     edge_linewidth=0.7,orig_dest_node_alpha=1, orig_dest_node_color='g')
```

Справа пример двух вершин, между которыми расстояние < 50.

ЗАДАНИЕ 1

Пункт 2

Определить, какой из объектов расположен так, что расстояние между ним и самым дальним домом минимально (“туда”, “обратно”, “туда и обратно”).

```python
def get_optimal_hospitals(short_path_matrix, hospital_values, apartments_values, new_nodes):

    optimal_hosps = []

    there = short_path_matrix.copy()
    back = short_path_matrix.copy()
    iter1 = 0
    iter2 = 0

    for i in range (len(new_nodes )):
      if new_nodes[i] not in hospital_values:
        #удаляем строку
        there = np.delete(there, iter1, 0)
        #удаляем столбец
        back = np.delete(back, iter1, 1))
        iter1 -= 1

      else:
        #удаляем столбец
        there = np.delete(there, iter2, 1)
        #удаляем строку
        back = np.delete(back, iter2, 0)

        iter2 -= 1

      iter1 += 1
      iter2 += 1

    # туда
    # возьмем минимум от максимума по столбцам
    index = (np.amax(there, 1)).argmin()
    optimal_hosps.append(apartments_values[index])

    # обратно
    # возьмем минимум от максимума по строкам
    index = (np.amax(back, 0)).argmin()
    optimal_hosps.append(apartments_values[index])

    # туда и обратно
    index = (np.amax(there, 1) + np.amax(back, 0)).argmin()
    optimal_hosps.append(apartments_values[index])
    return optimal_hosps
```

ЗАДАНИЕ 1

Пункт 2

```
opt = get_optimal_hospitals(data, hospital_values, apartments_values, new_nodes)
print(opt)
```

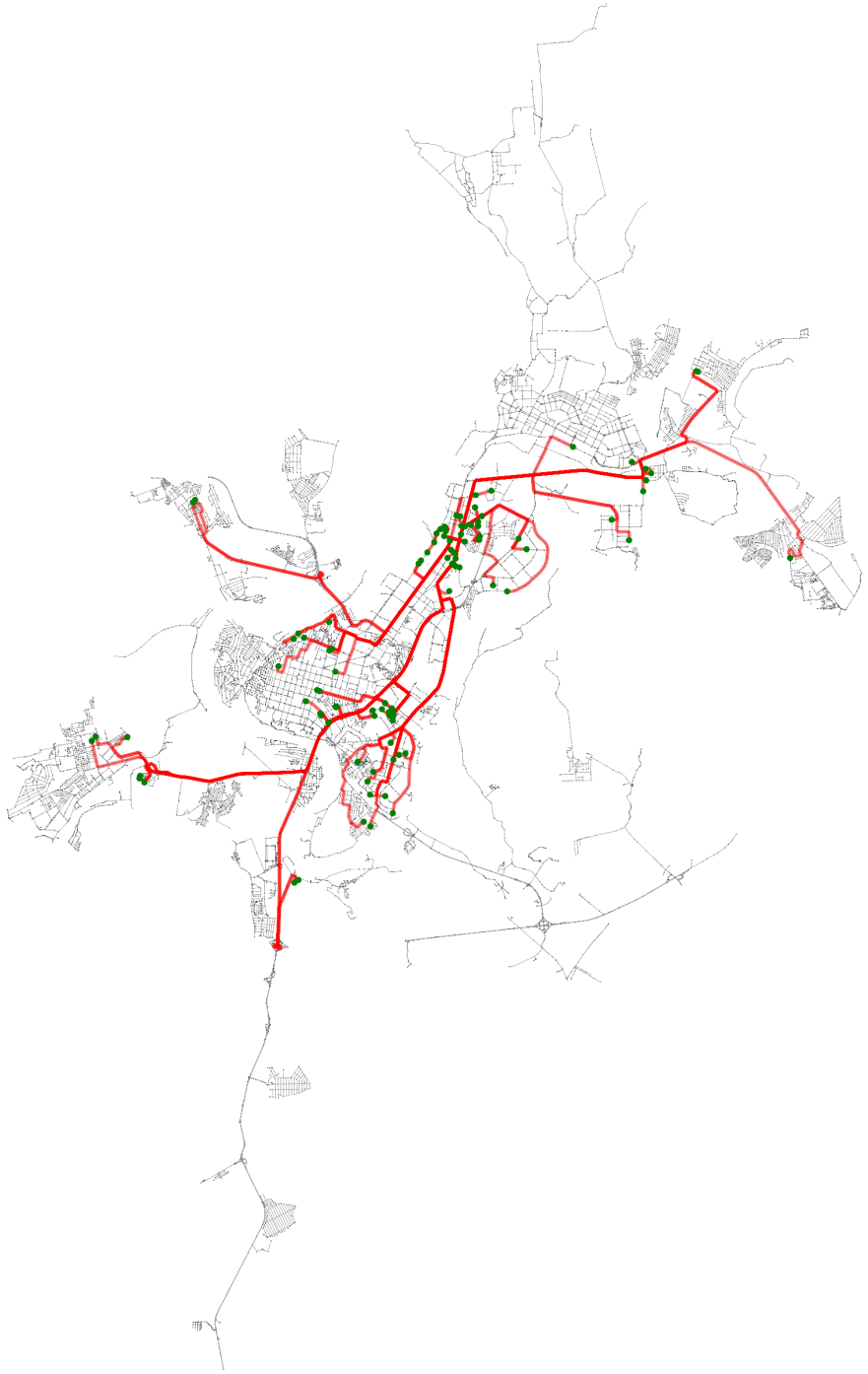
```
routes = []
for el in apartments_values:
  index = np.where(new_nodes == el)[0][0]
  route = paths[opt[2]][new_nodes[index]]
  routes.append(route)
```

ЗАДАНИЕ 1

Пункт 2

Определить, какой из объектов расположен так, что расстояние между ним и самым дальним домом минимально (“туда”, “обратно”, “туда и обратно”).

Визуализация данного дерева.



ЗАДАНИЕ 1

Пункт 3

Определить, для какого объекта инфраструктуры сумма кратчайших расстояний от него до всех домов минимальна.

```
def from_hosp_to_ap_min_sum (short_path_matrix, hospital_values, new_nodes):

    short_path_hosp_to_ap = short_path_matrix.copy()
    iter1 = 0
    iter2 = 0

    for i in range (len(new_nodes )):
      if new_nodes[i] not in hospital_values:
        #удаляем строку
        short_path_hosp_to_ap = np.delete(short_path_hosp_to_ap, iter1, 0)
        iter1 -= 1

      else:
        #удаляем столбец
        short_path_hosp_to_ap = np.delete(short_path_hosp_to_ap, iter2, 1)
        iter2 -= 1

      iter1 += 1
      iter2 += 1

    index = (np.sum(short_path_hosp_to_ap, 1)).argmin()
    return hospital_values[index]
```

```
min_hosp_id = from_hosp_to_ap_min_sum(data, hospital_values, new_nodes)
print(min_hosp_id)
```

```
2661495390
```

ЗАДАНИЕ 1

Пункт 4

Определить, для какого объекта инфраструктуры построенное дерево кратчайших путей имеет минимальный вес.

```python
def weight_reduced_tree(stack, nodes):
  weight = 0
  parent_must_use = False
  last_list = False
  while len(stack)>0:
    node = stack.pop()
    node_id, e, islist = node
    if islist:
      if node_id in nodes:
          weight += e
          parent_must_use = True
          last_list = True
      elif not last_list:
        parent_must_use = False
    else:
      if node_id in nodes:
          weight += e
          parent_must_use = True
          last_list = True
      elif parent_must_use:
          last_list = False
          weight += e
  return weight
```

```python
def tree_to_stack(stack, node, tree_adj):

      stack.append(node)
      node, _, islist = node
      if not islist:
        for child in tree_adj[node]:
            child_node, e = child
            if len(tree_adj[child_node]) > 0:
              tree_to_stack(stack, (child_node, e, False), tree_adj)
            else:
              tree_to_stack(stack, (child_node, e, True), tree_adj)
```

```python
weights_of_tree = np.zeros(M)

for i in range(M):
  hosp = hospital_values[i]
  stack = []
  tree_to_stack(stack, (hosp, 0, False), small_tree_dict_adj[hosp])
  weights_of_tree[i] = weight_reduced_tree(stack, apartments_values)

hospital_values[weights_of_tree.argmin()]
```

```
2661495390
```

ЗАДАНИЕ 2

В качестве метода кластеризации было предложено использовать метод полной связи (complete-linkage clustering).

Для этого нами была использована библиотека scipy.cluster.hierarchy. и методы linkage (method='complete’) (для выполнения второго пункта задания) и fcluster(Z, k, criterion = 'maxclust’) (для выполнения 3 и 4 пунктов).

Для построение дендрограммы также использовалась встроенная функция dendrogram().

ЗАДАНИЕ 2

Пункт 1

Построить дерево кратчайших путей от объекта до выбранных узлов. Вычислить общую длину дерева, а также сумму кратчайших расстояний от объекта до всех заданных узлов.

```
hosp_index = random.choice(hospital_values)
number = np.where (hospital_values == index)[0]
```

```
short_path_hosp_to_ap = short_path_matrix.copy()
iter1 = 0
iter2 = 0

for i in range (len(new_nodes )):
  if new_nodes[i] not in hospital_values:
    #удаляем строку
    short_path_hosp_to_ap = np.delete(short_path_hosp_to_ap, iter1, 0)
    iter1 -= 1

  else:
    #удаляем столбец
    short_path_hosp_to_ap = np.delete(short_path_hosp_to_ap, iter2, 1)
    iter2 -= 1

  iter1 += 1
  iter2 += 1
```

```
stack = []

tree_to_stack(stack, (index, 0, False), small_tree_dict_adj[hosp_index])

# общая длина дерева
weights_of_hosp = weight_reduced_tree(stack, apartments_values)

#сумма кратчайших путей

for i in range (N):
  sum_shortest_paths = sum(short_path_hosp_to_ap[number][0])
```

ЗАДАНИЕ 2

Пункт 1. Результаты.

```
hosp_index
```

```
3700686070
```

```
weights_of_hosp
```

```
7630
```

```
sum_shortest_paths
```

```
11001.0
```

Визуализация дерева кратчайших путей от объекта до выбранных узлов.

ЗАДАНИЕ 2

Пункт 2

Разбить выбранные узлы на кластеры, используя метод полной связи (complete-linkage clustering). Построить дендрограмму разбиения узлов.

```
our_clusters = linkage(ap_path_matrix, method='complete')
plt.figure()
```

```
set_link_color_palette(['m', 'c', 'y', 'k'])
fig, axes = plt.subplots(1, 2, figsize=(25, 10))
dn1 = dendrogram(our_clusters, ax=axes[0], above_threshold_color='y',
                 orientation='top')
dn2 = dendrogram(our_clusters, ax=axes[1], above_threshold_color='#bcbddc',
                 orientation='right')
plt.savefig('dendrogram', dpi=500, orientation='portrait', papertype=None,
            format=None, transparent=False, bbox_inches=None, pad_inches=0.1,
            frameon=None, metadata=None)

set_link_color_palette(None)  # reset to default after use
plt.show()
```

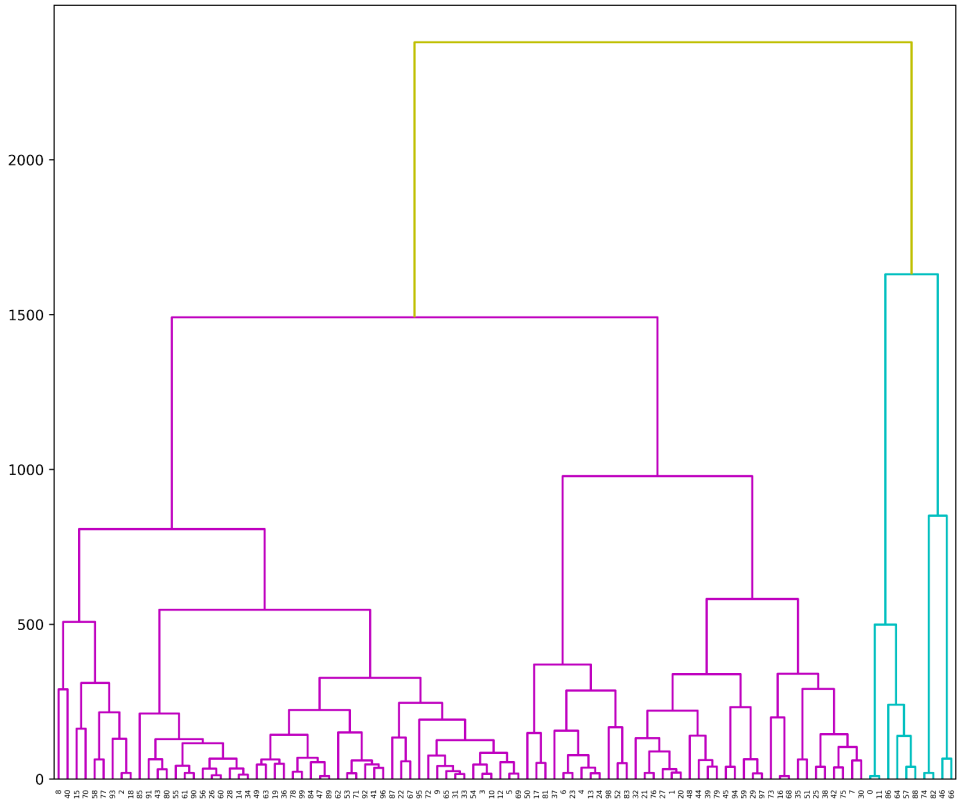
Зададим матрицу путей между выбранными узлами-домами.

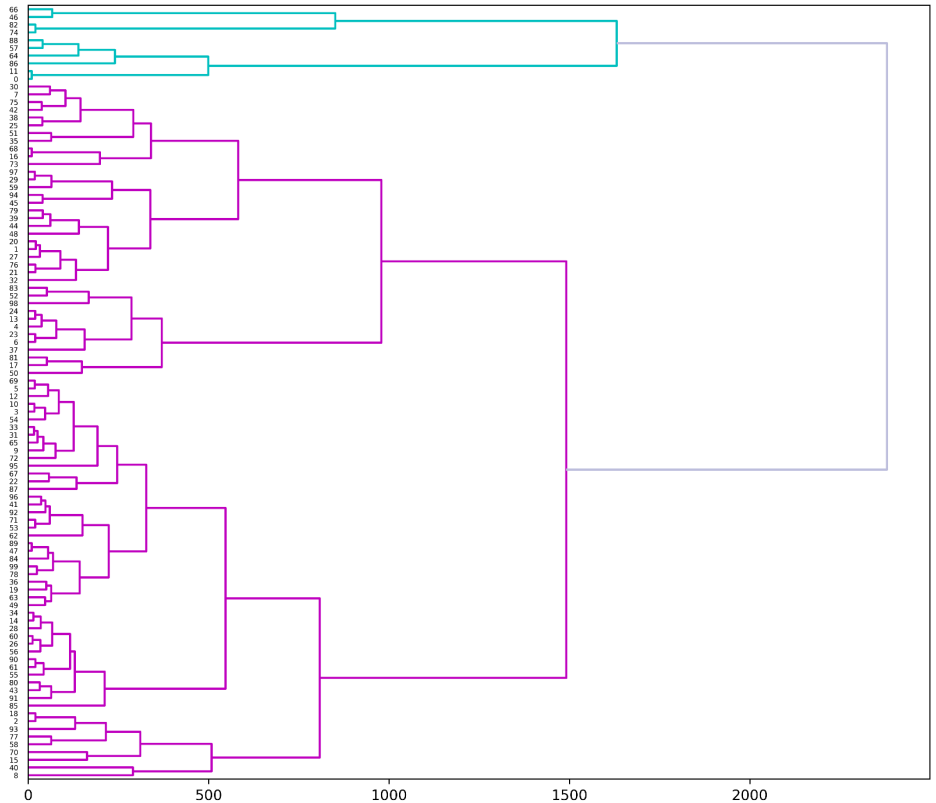
```
ap_path_matrix
```

```
array([[  0., 202., 272., ..., 218., 128., 289.],
       [204.,   0.,  87., ...,  52., 117., 115.],
       [270.,  92.,   0., ...,  83., 183.,  87.],
       ...,
       [219.,  50.,  80., ...,   0., 132., 132.],
       [127., 106., 176., ..., 122.,   0., 193.],
       [262., 108.,  87., ..., 123., 175.,   0.]])
```

ЗАДАНИЕ 2

Пункт 2. Полученная дендрограмма





ЗАДАНИЕ 2

Пункт 3

Пусть узлы разбиты на k кластеров.

- Найти расположение центра масс (центроида) для каждого кластера;
- Построить дерево кратчайших путей от объекта до центроидов.
- 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.

ЗАДАНИЕ 2

Для построения разбиения на k кластеров будем использовать уже упомянутую ранее функцию fclusters().

```
clusters = fcluster(our_clusters, k, criterion='maxclust')
```

В зависимости от k мы получаем различные результаты, здесь проиллюстрируем для k = 5.

```
array([2, 1, 1, 3, 2, 2, 1, 2, 1, 1, 2, 2, 2, 1, 2, 1, 2, 2, 1, 1, 1, 2,
       2, 2, 1, 2, 4, 2, 2, 2, 2, 1, 5, 2, 2, 2, 2, 4, 2, 1, 3, 2, 5, 2,
       2, 3, 5, 1, 3, 1, 1, 1, 1, 1, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 4, 2,
       4, 2, 2, 2, 2, 4, 1, 2, 2, 3, 2, 1, 1, 2, 2, 1, 1, 2, 2, 2, 2, 5,
       1, 2, 1, 1, 1, 2, 2, 3, 2, 5, 2, 1], dtype=int32)
```

```
plt.figure(figsize=(10, 8))
plt.scatter(path_matrix[:,0], path_matrix[:,1], c=clusters, cmap='prism')
plt.show()
```

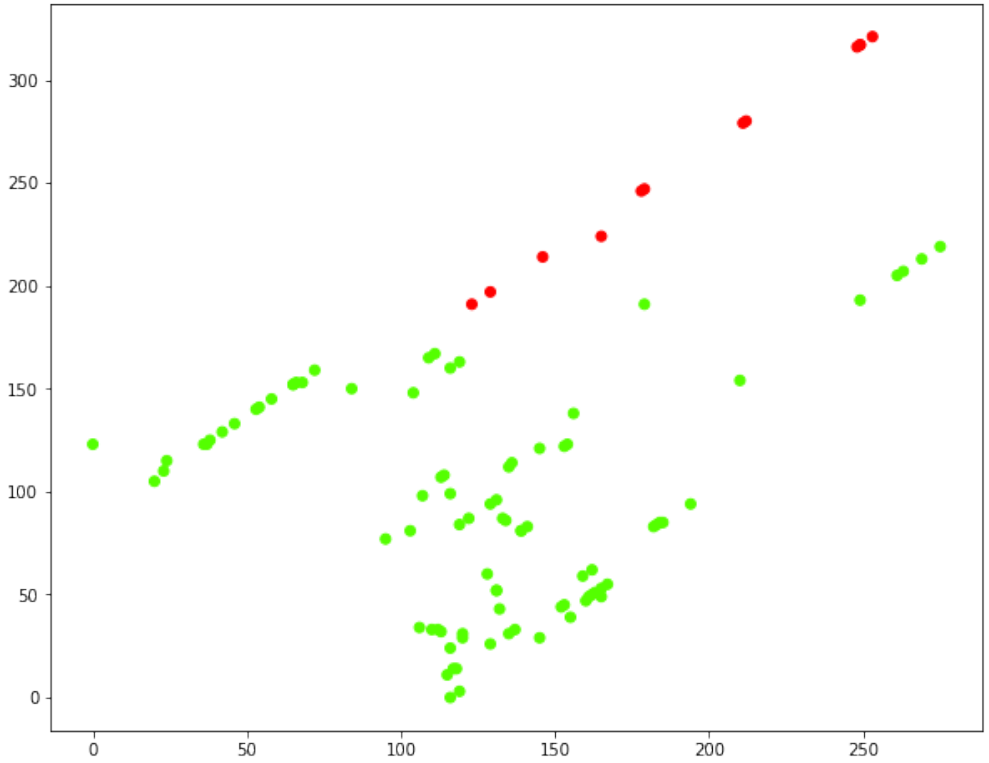
КЛАСТЕРЫ И ЦЕНТРОИДЫ

```
plt.figure(figsize=(10, 8))
plt.scatter(ap_path_matrix[:,0], ap_path_matrix[:,1], c=clusters, cmap='prism')
plt.savefig('clust5k', dpi=500, orientation='portrait', papertype=None,
            format=None, transparent=False, bbox_inches=None, pad_inches=0.1,
            frameon=None, metadata=None)
plt.show()
```

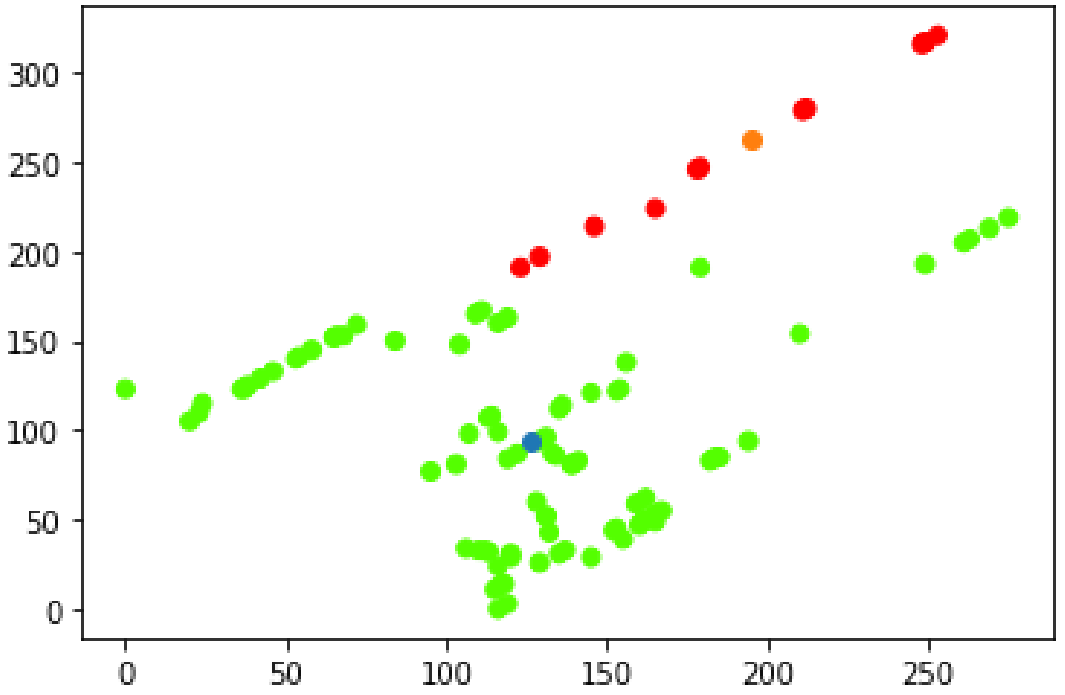
```
centroids_node = {}
centroids_node = centroids(ap_path_matrix, our_clusters, clusters, k, index )
```

```
def centroids(X, Z, clusters,  k):

    plt.figure()
    plt.scatter(X[:,0], X[:,1], c=clusters, cmap='prism')  # plot points with cluster depende

    clusters_us = {}
    clusters_us_id = {}
    for i in range(len(clusters)):
        j = clusters[i]
        if j in clusters_us:
            clusters_us[j].append(X[i])
            clusters_us_id[j].append(i)
        else:
            clusters_us[j] = [X[i]]
            clusters_us_id[j] = [i]

    for key, value in clusters_us.items():
        clusters_us[key] = np.array(value)

    centroids = {}
    centroids_node = {}
    for key, value in clusters_us.items():
        n = len(value)
        x = sum(value[:, 0]) / n
        y = sum(value[:, 1]) / n
        centroids[key] = [x, y]
        plt.scatter(x, y)

        e = 10000
        cent_node = 0
        for i in range(len(clusters_us_id[key])):
          dx = value[i, 0] - x
          dy = value[i, 1] - y
          e1 = ((dx+dy)**2)/2
          if (e1 < e):
            e = e1
            cent_node = i
        centroids_node[key] = clusters_us_id[key][cent_node]


    for key, value in clusters_us_id.items():
        n = len(value)
        x = 0
        y = 0

    print("Centroids")
    print(centroids)

    plt.savefig('centroids_in_clusters', dpi=500, orientation='portrait', papertype=None,
            format=None, transparent=False, bbox_inches=None, pad_inches=0.1,
            frameon=None, metadata=None)

    plt.show()
```

КЛАСТЕРЫ И ЦЕНТРОИДЫ, K=2

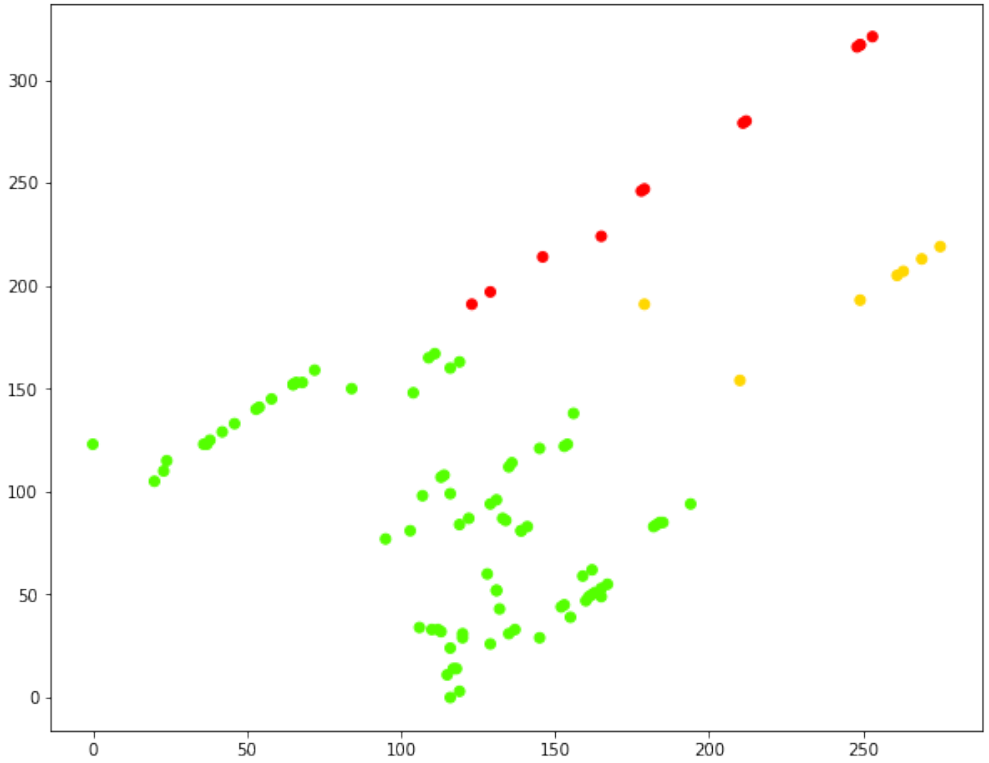


Кластеры

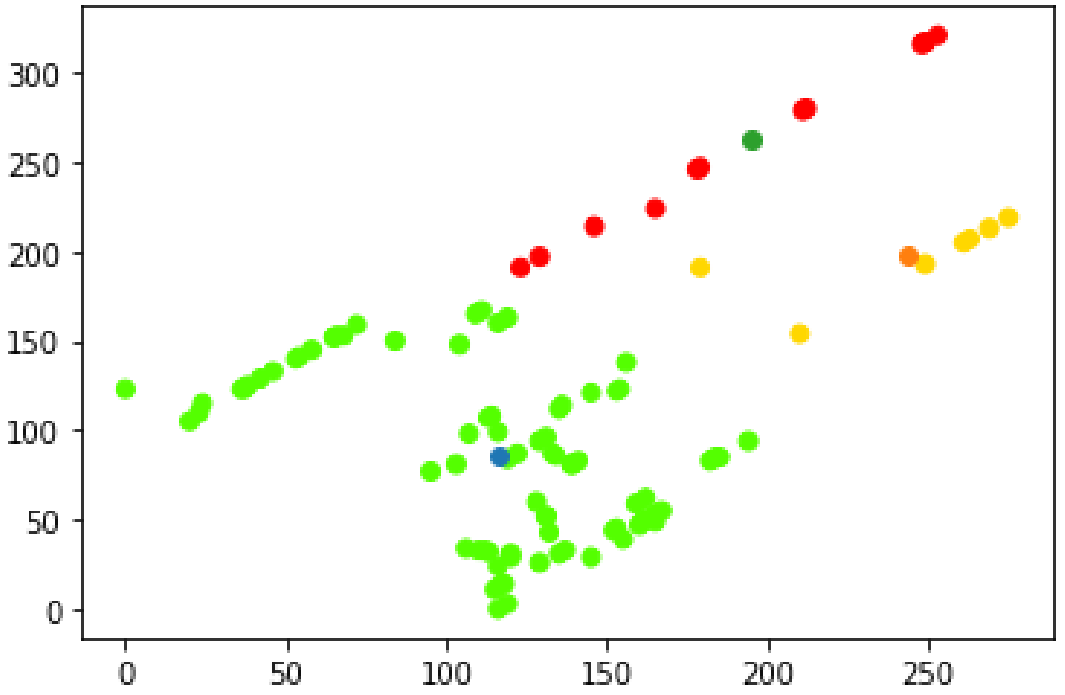


Центроиды в кластерах

КЛАСТЕРЫ И ЦЕНТРОИДЫ, K=3

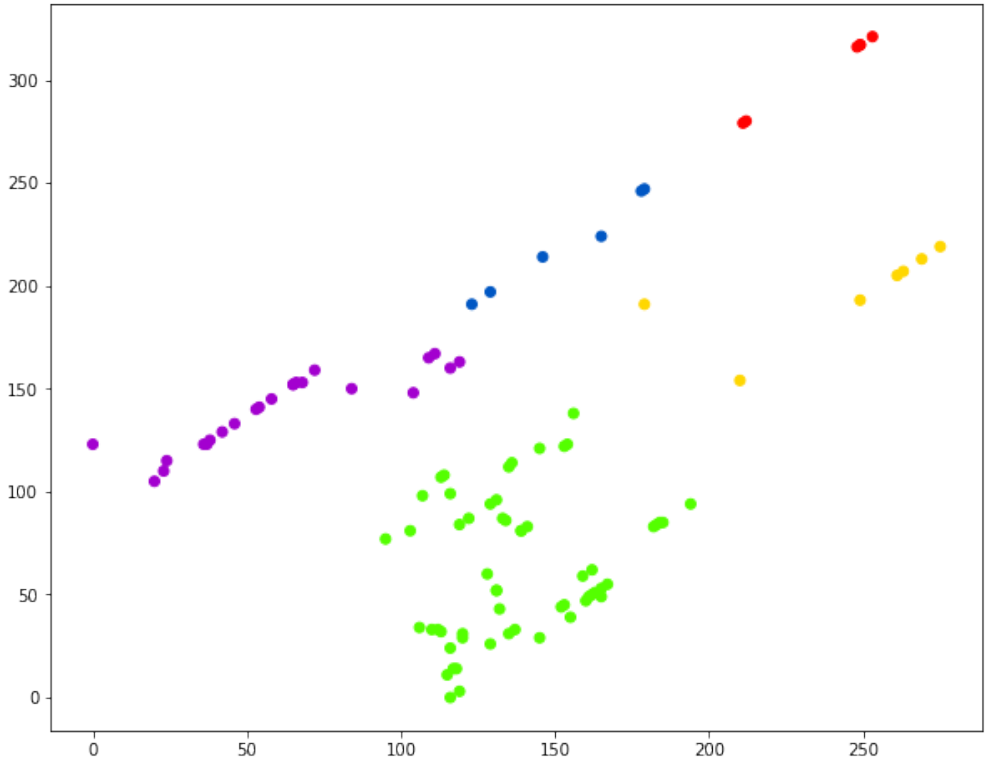


Кластеры

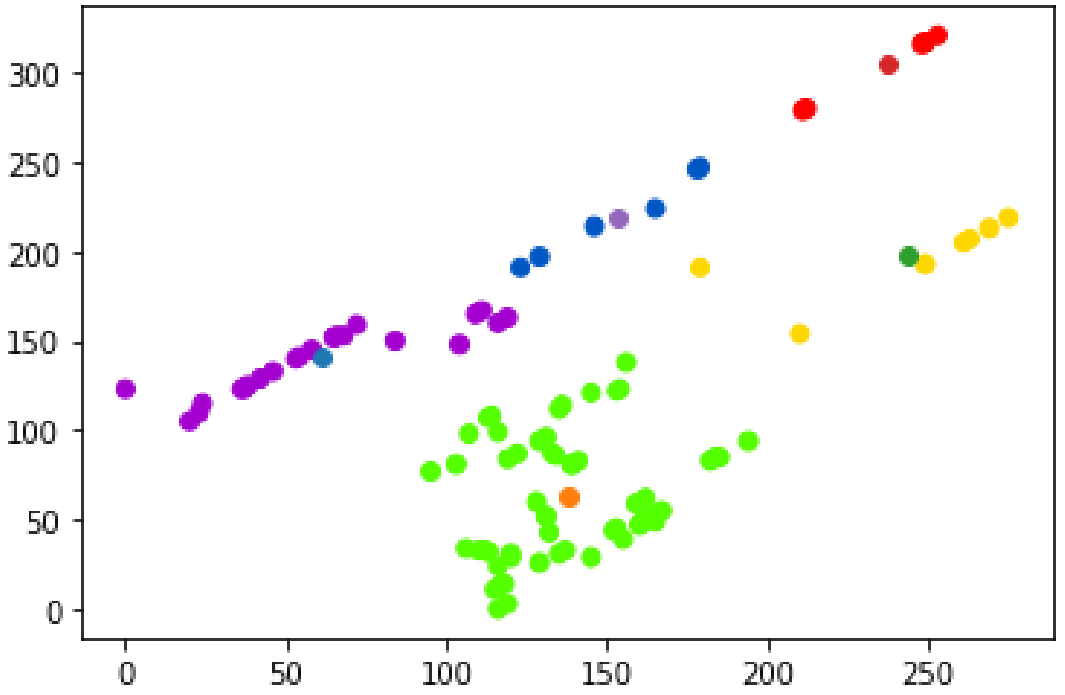


Центроиды в кластерах

КЛАСТЕРЫ И ЦЕНТРОИДЫ, K=5



Кластеры



Центроиды в кластерах

ПУТИ ОТ БОЛЬНИЦЫ ДО ЦЕНТРОИД

```
routes = []
for centr in centroids_node.values():
  route = paths[hosp_index][centr]
  routes.append(route)
```

```
ox.plot_graph_routes(G, routes, dpi=500, save=True, close=True, file_format='png',
                     filename='centroids_path_tree3k', fig_height = 30,
                     fig_width = 30, node_color='#000000', node_size = 1, orig_dest_node_size=50,
                     edge_color='#999999',  edge_linewidth=0.5, orig_dest_node_alpha=1,
                     orig_dest_node_color='g')
```

k=2

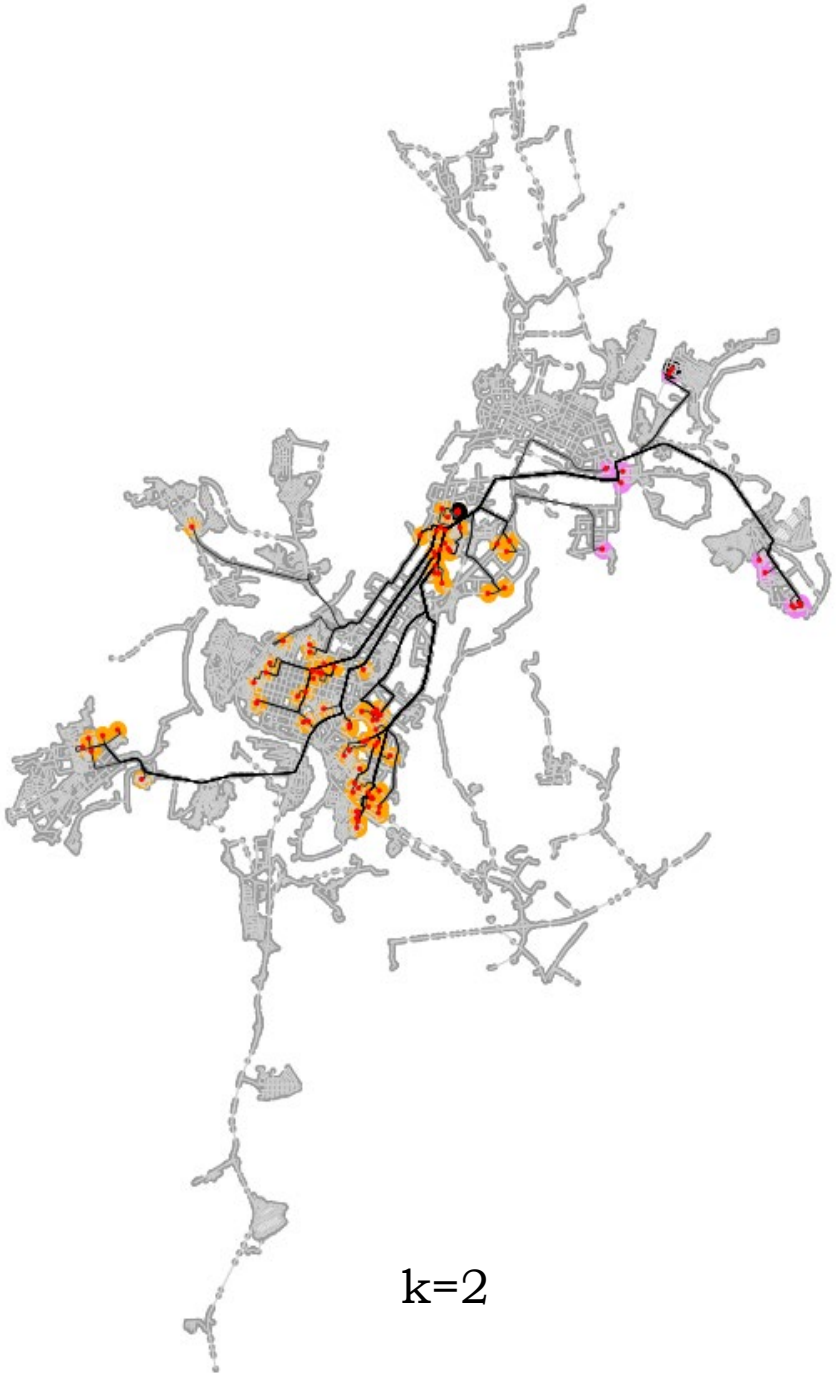
k=3

k=5

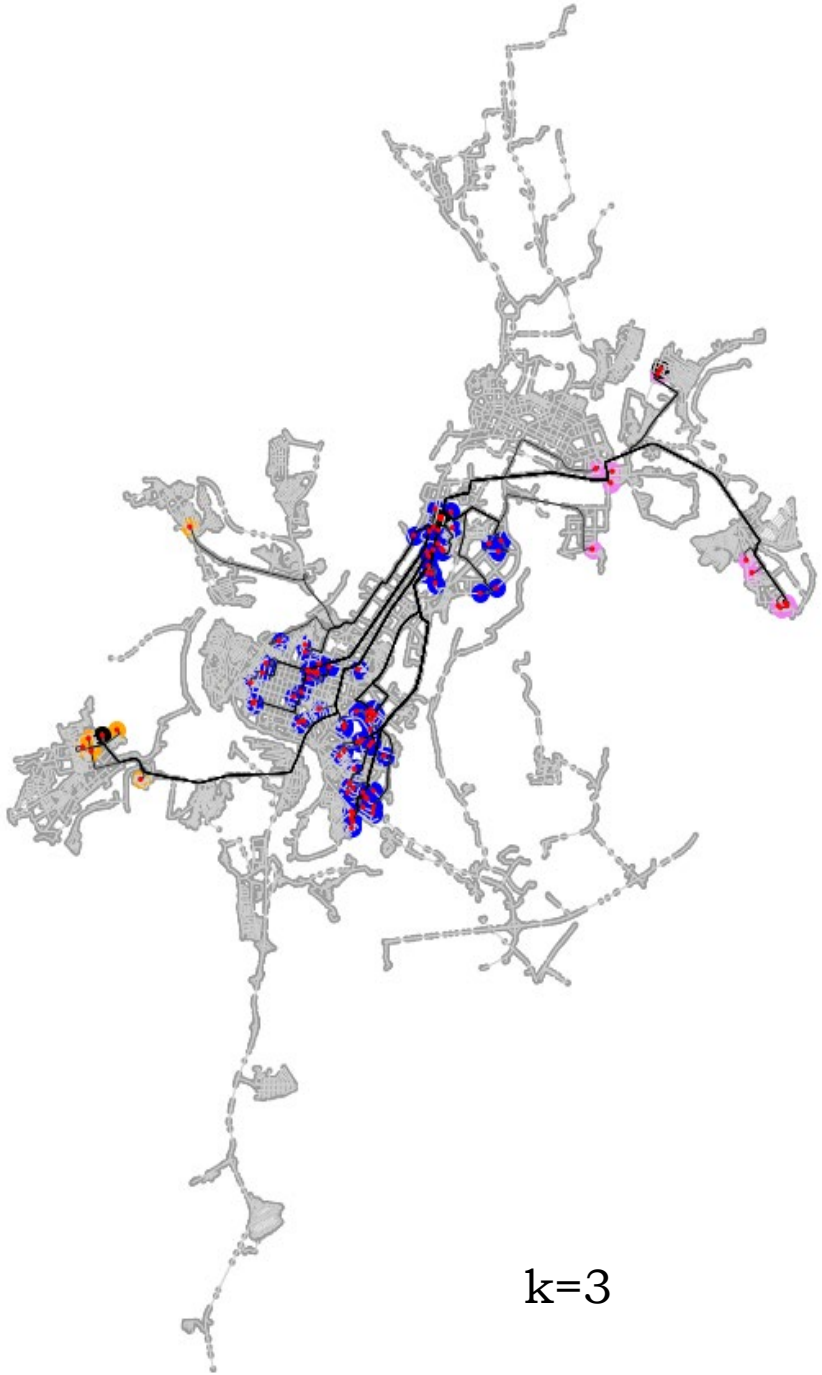
ПУТИ ОТ ЦЕНТРОИД ДО ОСТАЛЬНЫХ ЭЛЕМЕНТОВ КЛАСТЕРА

```
routes = []
for key, value in clusters_us_id.items():

    cent_node = centroids_node[key]
    start = nodes_numbers[cent_node]
    p, weight = dijkstra_heap(G, cent_node)


    for ap in apartments_values:
      try:
        route = nx.shortest_path(G, cent_node, ap, weight='length')
      except:
        route = [cent_node]
      routes.append(route)
```

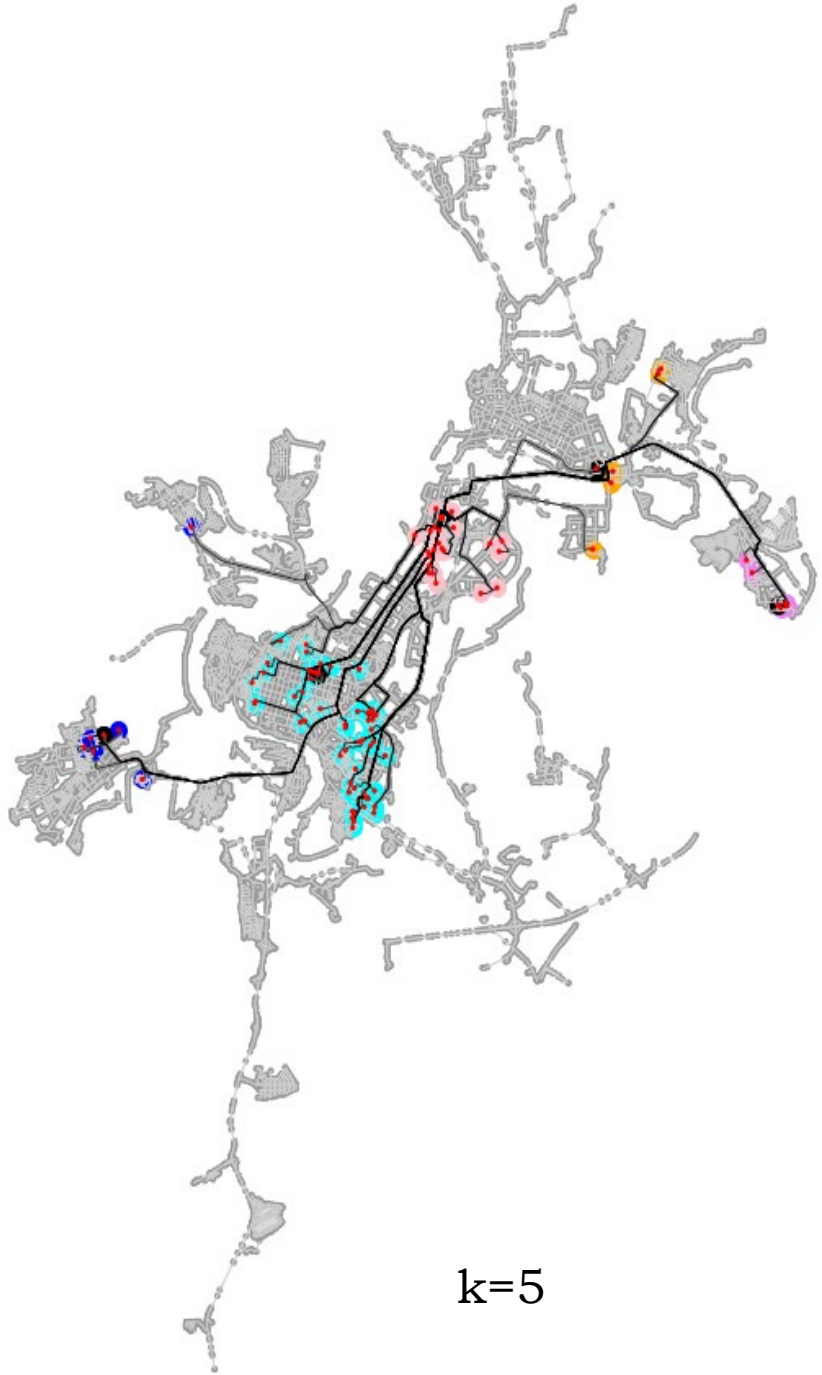
```
fig, ax = ox.plot_graph_routes(G, routes, fig_height = 30, fig_width = 30, show = False, close = False,
                               node_alpha=0, edge_color='lightgray', edge_alpha=1, edge_linewidth=0.8,
                               route_color='#00cc66', route_linewidth=0.8, route_alpha=1,
                               orig_dest_node_size=10, orig_dest_node_color='r', orig_dest_node_alpha=1)

for key in centroids_node:
  ax.scatter(G.nodes[centroids_node[key]]['x'], G.nodes[centroids_node[key]]['y'], c = 'black', s = 100)

for claster in range(1, k+1):
  if claster%2 == 0:
    color = 'orange'
  elif claster%3 == 0:
    color = 'blue'
  else:
    color = 'violet'

  for node in clusters_us_id[claster]:

    ap = apartments_values[node]
    ax.scatter(G.nodes[ap]['x'], G.nodes[ap]['y'], c = color, s = 100)

plt.savefig('short_path_centroid2k', dpi=500, orientation='portrait', papertype=None,
            format=None, transparent=False, bbox_inches=None, pad_inches=0.1,
            frameon=None, metadata=None)

plt.show()
```



k=2



k=3



k=5

ЗАДАНИЕ 2

Найти длину построенного дерева и сумму кратчайших расстояний от

объекта до всех заданных узлов.

```
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)
```

```python
def task23d(hosp_index, small_tree_dict_adj, centroids_node, apartments_values,
            short_path_hosp_to_ap, number, short_path_matrix):

    # найдем ребра от hosp_index до центроид

    weight_centroids_tree = 0

    stack = []
    tree_to_stack(stack, (hosp_index, 0, False), small_tree_dict_adj[hosp_index])
    weight_centroids_tree +=  weight_reduced_tree(stack, list(centroids_node.values()))

    apartments = []
    for ap in apartments_values:
      if not ap in centroids_node.values():
        apartments.append(ap)

    # найдем веса деревьев от центроид до остальных жилых домов

    for node in centroids_node.values():

        stack = []
        tree_to_stack(stack, (node, 0, False), small_tree_dict_adj[node])
        weight_centroids_tree += weight_reduced_tree(stack, apartments)

    sum_shortest_paths_centroids = 0

    for i in range (k):
      ind = np.where(apartments_values == centroids_node[i+1])
      sum_shortest_paths_centroids += short_path_hosp_to_ap[number, ind] * len(apartments)
      for ap in apartments:
        i1 = np.where(new_nodes == centroids_node[i+1])
        i2 = np.where(new_nodes == ap)
        sum_shortest_paths_centroids += short_path_matrix[i1, i2]

    return weight_centroids_tree, int(sum_shortest_paths_centroids)
```

ЗАДАНИЕ 2

Пункт 4

Сравнить найденные в п.1 и 3 величины для k = 2, 3, 5

Для выполнения задания пункт 2.3 был полностью запущен для различных значений параметра k. После были сделаны сравнения с результатами, полученными при выполнении пункта 2.1.

АНАЛИЗ РЕЗУЛЬТАТОВ КЛАСТЕРИЗАЦИИ

- Опираясь на теоретические знания и интуитивные соображения, проходя через центроиды мы должны были получить деревья веса меньшего, 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. Возможно, проблема в неправильном выборе центроид или подсчете веса. Или каких-то других неучтенных факторах.
- Несмотря на проблемы с итоговыми результатами, по изображениям можно сделать вывод: от выбранной точки оптимальным вариантом будет провести прямые дороги кратчайшего веса (насколько это возможно). Это даст возможность максимально быстро добраться до точки, откуда можно добраться до множества других узлов кластера за минимальное время.

ТЕСТИРОВАНИЕ ПРОГРАММЫ

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).

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

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

```
0.0001783370971679687 5
```

Запуск на матрице 3 на 3.
Алгоритм Дейкстры был запущен из 3 вершин, результаты записаны в csv-файл.

```
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]])
```

```
0.0013263225555419922
```

Запуск на матрице 5 на 5.
Алгоритм Дейкстры был запущен из 5 вершин, результаты записаны в csv-файл.

```
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]])
```

```
0.0012493133544921875
```

Запуск на матрице 8 на 8.
Алгоритм Дейкстры был запущен из 8 вершин, результаты записаны в csv-файл.

РЕЗУЛЬТАТЫ ТЕСТОВ

```
array([[  0,   2,   5, ..., 116, 125, 129],
       [  2,   0,   3, ..., 114, 123, 127],
       [  5,   3,   0, ..., 111, 120, 124],
       ...,
       [116, 114, 111, ...,   0,   9,  13],
       [125, 123, 120, ...,   9,   0,   4],
       [129, 127, 124, ...,  13,   4,   0]])
```

```
0.9804565906524658
```

Запуск на матрице 400 на 400. Алгоритм Дейкстры был запущен из всех 400 вершин, результаты записаны в csv-файл.

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

```
0.18947148323059082
```

Запуск на матрице 10000 на 10000. К сожалению, не удалось построить матрицу кратчайших расстояний от всех точек (лимит по RAM был достигнут при запуске алгоритма Дейкстры из 1000 вершин до 10000). Поэтому, как пример, приводится «строка» расстояний из первой (нулевой) вершины до всех 10000, результаты записаны в csv-файл.

СПАСИБО ЗА ВНИМАНИЕ!

GitHub проекта:
https://github.com/oksanamda/graphs