练习 4

1、问题描述

已知 95 个目标点的数据见 Excel 文件 data6.xlsx，第 1 列是这 95 个点的编号，第 2,3 列是这 95 个点的 x, y 坐标，第 4 列是这些点重要性分类，标明“1”的是第一类重要目标点，标明“2”的是第二类重要目标点，未标明类别的是一般目标点，第 5，6，7 标明了这些点的连接关系。如第三行的数据

C -1160 587.5 D F

表示顶点 C 的坐标为 $(-1160, 587.5)$，它是一般目标点，C 点和 D 点相连，C 点也和 F 点相连。

研究如下问题：

（1）画出上面的无向图，一类重要目标点用“五角星”画出，二类重要点用“*”画出，一般目标点用“.”画出。

要求必须画出无向图的度量图，顶点的位置坐标必须准确，不要画出无向图的拓扑图。

（2）当权重为距离时，求上面无向图的最小生成树，并画出最小生成树。

（3）求顶点 L 到顶点 R3 的最短距离及最短路径，并画出最短路径。

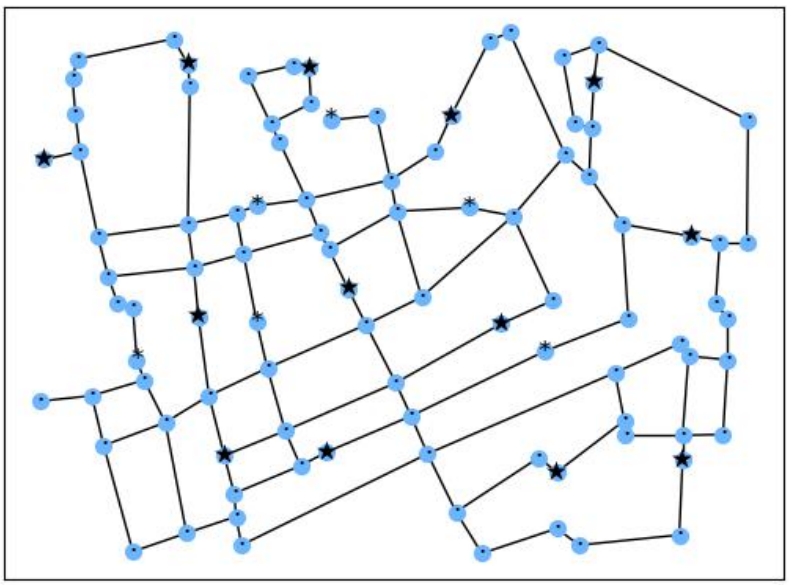
2、无向图

（1）read_excel 导入数据；

（2）按照格式读取坐标、点；

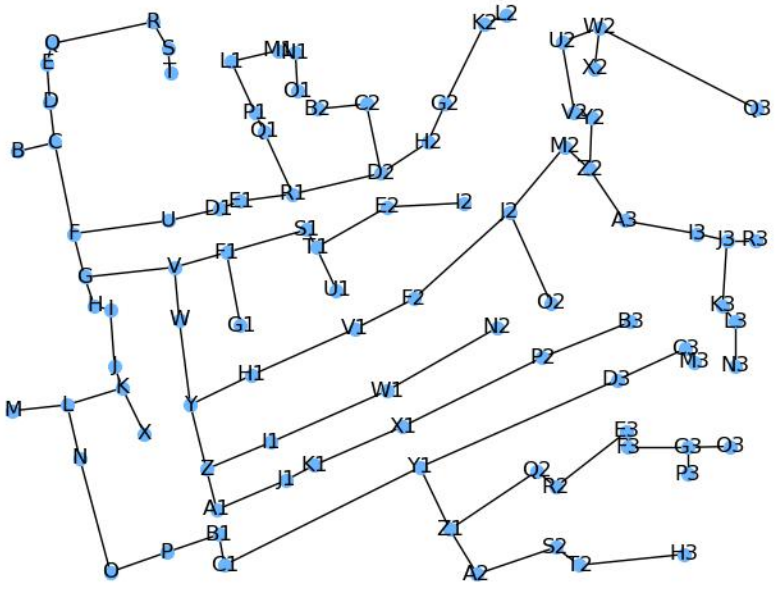
（3）导入边、获取映射（用字典存储）；

（4）绘制；



3、最小生成树

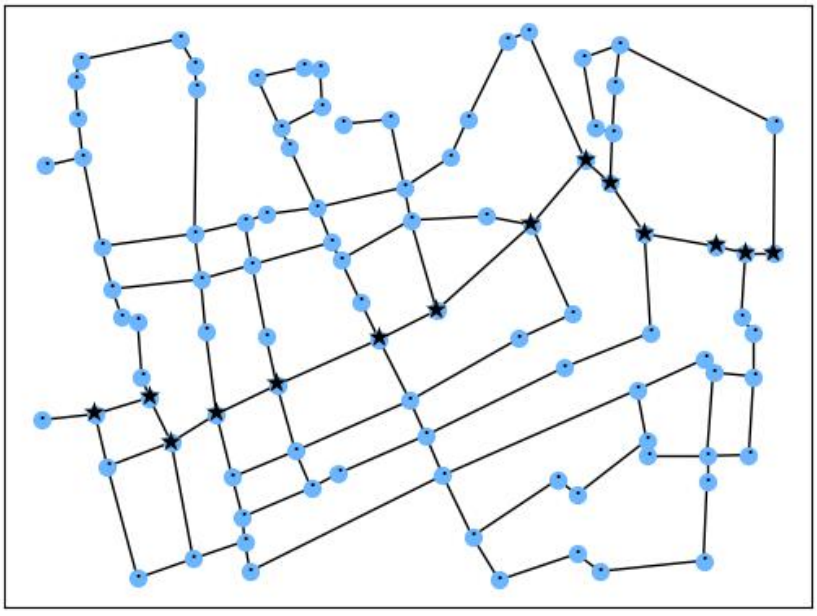
（1）minimum_spanning_tree 生成最小生成树；
（2）to_numpy_matrix 获得邻接矩阵，并计算权重；边的权重即为坐标的欧式距离；
（3）绘制：



最小生成树权重： 19145.88554983767

4、最短路径

（1）在第一问的基础上，构造加权边：G.add_weighted_edges_from(edges)；
（2）用 dijkstra_path 得到最短路径；
（3）绘图：



dijkstra 方法寻找最短路径：
节点 L 到 R3 的路径： ['L', 'K', 'X', 'Y', 'H1', 'V1', 'F2', 'J2', 'M2', 'Z2', 'A3', 'I3', 'J3', 'R3']
dijkstra 方法寻找最短距离：
节点 L 到 R3 的距离为： 2795.4679098753263

5、代码

```
import networkx as nx
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# 导入数据
data = pd.read_excel('data6.xlsx', header=None)
data = data.values[1:]
G = nx.Graph()

def draw():
    # 得到坐标
    vnode = data[...,1:3]
    # 得到点并导入
    nodes = list(data[...,0])
    G.add_nodes_from(nodes)
    # 导入边
    edges = []
    for l in data:
        n1 = l[0]
        for n2 in l[4:]:
            if n2 is not np.nan:
                edges.append([n1,n2])
    G.add_edges_from(edges)
    npos = dict(zip(nodes, vnode))  # 获取节点与坐标之间的映射关系，用字典表示
    # 导入标志
    # 绘制五角星
    # ax = plt.figure()
    labels = []
    for l in data[...,3]:
        if l==1:
            labels.append('★')
        elif l==2:
            labels.append('*')
        else:
            labels.append('·')
    nlabels = dict(zip(nodes, labels))  # 标志字典，构建节点与标识点之间的关系
    nx.draw_networkx_nodes(G, npos, node_size=50, node_color="#6CB6FF")  # 绘制节点
```

```
    nx.draw_networkx_edges(G, npos, edges)  # 绘制边
    nx.draw_networkx_labels(G, npos, nlabels)  # 标签
    # nx.draw_networkx(G)
    plt.show()

def distant(vnode, i, j):
    return np.sqrt(np.square(vnode[i,0]-vnode[j,0]) + np.square(vnode[i,1]-vnode[j,1]))

# def minTree():
def minTree():
    # 得到坐标
    vnode = data[...,1:3]
    # 得到点并导入
    nodes = list(data[...,0])
    # 获取节点与坐标之间的映射关系，用字典表示
    npos = dict(zip(nodes, vnode))
    # 获得最小生成树
    T = nx.minimum_spanning_tree(G)
    # 得到邻接矩阵
    C = nx.to_numpy_matrix(T)
    # 计算两点距离
    for i in range(95):
        for j in range(95):
            if C[i,j] == 1:
                C[i,j] = distant(vnode, i, j)
    w = C.sum()/2
    print("最先生成树的权重 W=\n", w)
    # 画图
    nx.draw(T, npos, with_labels=True, node_size=50, node_color="#6CB6FF")
    w2 = nx.get_edge_attributes(T, 'weight')
    nx.draw_networkx_edge_labels(T, npos, edge_labels=w2)
    plt.show()

def path():
    # 得到坐标
    vnode = data[...,1:3]
    # 得到点并导入
    nodes = list(data[...,0])
    # 节点名映射到 int ，方便计算距离
    nodes2int = {}
    for i in range(len(nodes)):
        nodes2int[nodes[i]] = i
    # 导入点
    G.add_nodes_from(nodes)
```

```
    # 导入边
    edges = []
    for l in data:
        n1 = l[0]
        for n2 in l[4:]:
            if n2 is not np.nan:
                edges.append([n1,n2,distant(vnode, nodes2int[n1], nodes2int[n2])])
    G.add_weighted_edges_from(edges)
    # 映射坐标
    npos = dict(zip(nodes, vnode))  # 获取节点与坐标之间的映射关系，用字典表示
    # 计算最短路径
    print('dijkstra 方法寻找最短路径：')
    path = nx.dijkstra_path(G, source='L', target='R3')
    print('节点 L 到 R3 的路径：', path)
    print('dijkstra 方法寻找最短距离：')
    distance = nx.dijkstra_path_length(G, source='L', target='R3')
    print('节点 L 到 R3 的距离为：', distance)
    # 画出最短路径
    labels = []
    for l in data[...,0]:
        if l in path:
            labels.append('★')
        else:
            labels.append('·')
    nlabels = dict(zip(nodes, labels))  # 标志字典，构建节点与标识点之间的关系
    nx.draw_networkx_nodes(G, npos, node_size=50, node_color="#6CB6FF")  # 绘制节点
    nx.draw_networkx_edges(G, npos, edges)  # 绘制边
    nx.draw_networkx_labels(G, npos, nlabels)  # 标签
    plt.show()

if __name__ == '__main__':
    draw()
    minTree()
    path()
```