Week4

19377419 孙启航

作业要求

> 图是非常重要的一种数据结构，在社交媒体和现实生活中这一结构非常常见，且有大量分析统计基于图结构进行。本次作业提供了newmovies数据集（见资源/data/newmovies.txt），希望基于该数据，在程序中读取并存储用户节点信息，建立无向图结构，并进一步实现相关统计和可视化功能。
>
> 1. newmovies.txt保存了相关数据，其中*Vertices 34282 下的每一行为一个节点，表示一位明星、编剧或电影。每一行中属性以\t分割，分别为节点id，名称，节点权重，节点类型，其他信息（其他信息以";"分割)。
>   注意，节点里的权重信息是原数据集提供的，本次作业用不到，另外edges部分的参数每行三个数，前两个是边所连接的节点id，第三个值均为1。
> 2. 建立包GraphStat，实现相关网络的构建和可视化。其中
>   1. 包Graph，用以实现点和图结构的创建，以及相关的基础统计功能
>   1. 实现node.py模块
>   1. 实现函数init_node()，输入相关的节点文件，返回对应的节点列表，其中每个元素为一个节点
>   2. 实现函数print_node()，利用format函数，将节点属性输出至屏幕上
>   2. 对graph.py模块，同上，同时实现图结构的序列化存储。
>   3. 实现stat.py模块，基于图结构，进行基础的统计
>   1. 计算图中的平均度并返回
>   2. 统计图中节点某个属性的分布情况
>   2. 包Visualization， 基于上述构建的图和节点结构，利用matplotlib等绘制相关的统计结果
>   1. plotgraph.py
>   绘制基于图的信息，如图的结构
>   2. plotnodes.py
>   绘制节点的属性分布，并提供结果保存

主要代码

测试代码

```python
import GraphStat.NetworkBulider.node as GsNbNode
import GraphStat.NetworkBulider.graph as GsNbGraph
import GraphStat.NetworkBulider.stat as GsNbStat
import GraphStat.Visualization.plotgraph as GsVPlot
import GraphStat.Visualization.plotnodes as GsVPlotNds


def test_node_module(lines):

    NodeInfoList = GsNbNode.init_node(lines)
    # 返回一个列表组成的矩阵

```

```
    GsNbNode.print_node(NodeInfoList[1])
    # 打印第二个节点的信息
    # *Node Infomation*
    # ID:                       1
    # Name:                     "Karen Allen"
    # Weight:                   7467
    # Node Type:                starring

    # *Other Information*:
    # American film actors
    # American stage actors
    # American video game actors
    # Bard College at Simon's Rock faculty
    # Illinois actors
    # People from Greene County, Illinois
    # Saturn Award winners

    # Adjecent Table:
    # [[1, 11814, 1], [1, 3869, 1], [1, 11348, 1], [1, 9023, 1], [1, 7353, 1],
[1, 11190, 1], [1, 13214, 1], [1, 4562, 1], [1, 5465, 1], [1, 6106, 1], [1,
7922, 1], [1, 13175, 1], [1, 11814, 1]]

    degree = GsNbNode.get_degree(NodeInfoList[1])
    # The degree of the vertex is 13.
    # 但是这里仅仅统计的是从该节点出发的边，因为该图是无向图，所以真实的节点的度可能大于13
    AdjacentNodeList = GsNbNode.get_adjecent_node(NodeInfoList[1])
    # 这里也仅仅统计的是从节点1出发的边相邻的点，因为该图是无向图，所以真实的节点1相邻的节点并
不止这些
    # The Adjacent node of the 1 node is:
    # 11814
    # 3869
    # 11348
    # 9023
    # 7353
    # 11190
    # 13214
    # 4562
    # 5465
    # 6106
    # 7922
    # 13175

    return NodeInfoList


def test_graph_module(lines):

    # 返回一个字典，分别存储节点信息和边信息
    # 利用邻接表结构进行无向图的存储
    Graph_list = GsNbGraph.init_graph(lines)

    # 序列化图信息
    GsNbGraph.save_graph(Graph_list)

    # 重新加载已经序列化存储的信息
    load_list = GsNbGraph.load_graph("BUAA_21/Week4/graph_info.json")

    return Graph_list
```

```python
    #return Graph_list


def test_stat_module(graph):

    average_degree = GsNbStat.cal_average_degree(graph)
    print("\nThe average degree of the graph is:%.2f.\n" % average_degree)
    # The average degree of the graph is: 7.16.

    # 计算网络的度分布，返回一个字典
    degree_distribution_dict = GsNbStat.cal_degree_distribution(graph)
    # print("The degree distribution dict is:\n", degree_distribution_dict)
    # The degree distribution dict is:
    #  {'0': 6, '1': 12, '2': 13, '3': 5, '4': 0, '5': 1, '6': 4, '7': 1, '8':
0, '9': 0, '10': 1, '11': 1, '12': 1, '13': 4, '14': 1, '15': 7, '16': 0, '17':
3, '18': 1, '19': 1, '20': 10, '21': 2, '22': 3, '23': 0, '24': 14, '25': 0,
'26': 0, '27': 1, '28': 4, '29': 4, '30': 0, '31': 5, '32': 0, '33': 0, '34':
21, '35': 0, '36': 24, '37': 2, '38':
    # 1, '39': 4, '40': 8, '41': 1, '42': 1, '43': 4, '44': 0, '45': 1, '46': 2,
'47': 1, '48': 1, '49': 4, '50': 0, '51': 2, '52': 4, '53': 3, '54': 1, '55': 1,
'56': 5, '57': 2, '58': 11, '59': 33, '60': 1, '61': 2, '62': 1, '63': 0......}

    # 根据存储的图的结构，得到每个节点的类型，返回一个字典
    type_dict = GsNbStat.get_type(graph)
    # print("The type distribution of the graph is:\n", type_dict)
    #  The type distribution of the graph is:
    #  {'0': 'starring', '1': 'starring', '2': 'writer', '3': 'director', '4':
'starring', '5': 'starring', '6': 'starring', '7': 'director', '8': 'writer',
'9': 'writer', '10': 'starring', '11': 'director', '12': 'writer', '13':
'starring', '14': 'starring', '15': 'writer', '16': 'director', '17': 'writer',
'18': 'starring', '19': 'starring', '20':
    # 'starring', '21': 'writer', '22': 'director', '23': 'director',...}


def test_plotgraph_module(lines):
    # 度在[x_1, x_2]范围内的的分布图
    # 输入节点文件即可
    # 生成度在1~20之间的节点的频数分布
    GsVPlot.plotdegree_distribution(lines, 1, 20)

    # 绘制节点文件的网络图
    # GsVPlot.draw_graph(lines)
    # 运行时间超过20min，还是没有生成结果


def test_plotnode_module(lines):
    # 生成所有结点的type的统计结果，并绘制柱状图
    GsVPlotNds.plot_nodes_attr(lines, 'type')


def data_clean():
    '''
    清洗数据
    去除重复的边
    '''
    with open("BUAA_21/Week4/newmovies.txt", 'r', encoding='utf-8') as f:
        lines = f.readlines()
    edge_index = lines.index('*Edges\n')
```

```
    new = []
    new.extend(lines[:edge_index])

    for i in lines[edge_index+1:]:
        lis = i.split('\t')[0:2]
        lis = sorted(lis, key=int)
        if lis not in new:
            new.append(lis)
        else:
            pass
    with open("BUAA_21/Week4/newnew_movies.txt", 'w') as f:
        for i in new[:edge_index]:
            print(i, end='', file=f)
        print("*Edges\n", end='', file=f)
        for i in new[edge_index:]:
            print("%s\t%s\t1" % (i[0], i[1]), file=f)


if __name__ == '__main__':

    file = "C:/Users/DELL/Desktop/Code/BUAA_21/Week4/newnew_movies.txt"

    # data_clean()

    with open(file, 'r', encoding='utf-8') as f:
        lines = f.readlines()

    with open("BUAA_21/Week4/NodeInfoList.txt", "w+") as f:
        NodeInfoList = test_node_module(lines)
        print(*NodeInfoList, sep='\n', file=f)

    with open("BUAA_21/Week4/graph.txt", "w+") as f:
        graph = test_graph_module(lines)
        print(*graph, sep='\n', file=f)

    # 是个测试模块，并不需要返回结果，如果有返回变量的需要直接调用stat.py模块中的函数即可
    test_stat_module(graph)

    test_plotgraph_module(lines)

    test_plotnode_module(lines)
```

GraphStat包代码

__init__.py

```
if __name__ == 'main':
    print("作为主程序运行")
else:
    print("GraphStat初始化")
```

NetworkBuilder

graph.py

```
import pickle
```

```python
def init_graph(lines):
    '''

    返回一个字典，分别存储节点信息和边信息
    利用邻接表结构进行无向图的存储
    '''
    index_egde = lines.index("*Edges\n")

    # # 只保留存储节点的信息
    # # print(index_egde)
    # # 创建一个空矩阵，存储所有结点的信息
    # NodeInfoList = []
    Graph_List = []

    for line in lines[1:index_egde]:
        dic = {}
        vector = {}
        lis = line.split('\t')

        dic['Id'] = lis[0]
        dic['Name'] = lis[1]
        dic['Weight'] = lis[2]  # 因为权重信息并不用得上，所以加入字典中并不影响
        dic['Type'] = lis[3]
        dic['OtherInfo'] = lis[4].split('\n')[0].split(';')[:-1]
        # dic['EdgesTable'] = []

        vector["Vertex"] = dic
        vector["AdjacentNode"] = []
        Graph_List.append(vector)

    #     # {'Id': '34282', 'Name': '"The Lonesome Mouse"', 'Type': 'movie',
'OtherInfo': ['1943 films', 'Tom and Jerry cartoons']}
    # # 之后加入邻接表信息
    for line in lines[index_egde+1:]:
        lis = line.split('\t')
        if int(lis[1]) not in Graph_List[int(lis[0])]["AdjacentNode"]:
            Graph_List[int(lis[0])]["AdjacentNode"].append(int(lis[1]))
        else:
            pass

    return Graph_List


def save_graph(graph):
    '''
    序列化图信息
    '''
    with open("BUAA_21/Week4/graph_info.json", "wb") as f:
        pickle.dump(graph, f)


def load_graph(graph_info):
    '''
    重新加载已经序列化存储的信息
    '''
    with open(graph_info, "rb") as f:
        info = pickle.load(f)
```

```
    return info
```

node.py

```
def init_node(lines):
    '''
    返回字典dic，key为节点的属性，值为对应的属性值
    '''

    index_egde = lines.index("*Edges\n")

    # 只保留存储节点的信息
    # print(index_egde)
    # 创建一个空矩阵，存储所有结点的信息
    NodeInfoList = []

    for line in lines[1:index_egde]:
        dic = {}
        lis = line.split('\t')

        dic['Id'] = lis[0]
        dic['Name'] = lis[1]
        dic['Weight'] = lis[2]  # 因为权重信息并不用得上，所以加入字典中并不影响
        dic['Type'] = lis[3]
        dic['OtherInfo'] = lis[4].split('\n')[0].split(';')[:-1]
        dic['EdgesTable'] = []
        NodeInfoList.append(dic)
        # {'Id': '34282', 'Name': '"The Lonesome Mouse"', 'Type': 'movie',
'OtherInfo': ['1943 films', 'Tom and Jerry cartoons']}
    # 之后加入该节点相邻的边的信息
    for line in lines[index_egde+1:]:
        lis = line.split('\t')
        EdgesVector = [int(lis[0]), int(lis[1]), 1]
        if EdgesVector not in NodeInfoList[int(lis[0])]["EdgesTable"]:
            NodeInfoList[int(lis[0])]['EdgesTable'].append(EdgesVector)
        else:
            pass
        # NodeInfoList[int(lis[0])]['EdgesTable'].append(EdgesVector)

    #  {'Id': '22', 'Name': '"Bruce Malmuth"', 'Type': 'director', 'OtherInfo':
['1934 births', '2005 deaths', 'American film directors', 'Deaths from throat
cancer', 'Cancer deaths in California'], 'EdgesTable': [[22, 13393, 1], [22,
837, 1], [22, 242, 1]]}
    return NodeInfoList


def get_degree(node):
    '''
    获取对应的节点的度
    '''
    degree = len(node["EdgesTable"])

    print("The degree of the vertex is %d." % degree)


def get_adjecent_node(node):
```

```
    '''
    获取该节点相邻的节点，按行打印，并返回一个列表
    '''

    AdjacentNodeList = []
    print("\nThe Adjacent node of the %s node is:" % node["Id"])
    for item in node["EdgesTable"]:
        AdjacentNodeList.append(item[1])
        print(item[1])
    return AdjacentNodeList


def print_node(node):
    '''
    显示节点全部信息（利用format函数）
    '''
    print("\n*Node Infomation*\nID:\t\t\t{:<8s}\nName:\t\t\t{:
<20s}\nWeight:\t\t\t{:<10s}\nNode Type:\t\t{:<15s}\n\n*Other
Information*:\n{}\n\nEdges Table:\n{}".format(
        node["Id"], node["Name"], node["Weight"], node["Type"],
"\n".join(node["OtherInfo"]), node["EdgesTable"]))
    return None
```

stat.py

```
def cal_average_degree(graph):
    '''
    计算网络中的平均度
    '''

    adjacent_table = []
    sum_degree = 0
    for i in graph:
        vector = []
        vector.append(int(i['Vertex']['Id']))
        vector.append(i['AdjacentNode'])

        adjacent_table.append(vector)

        sum_degree += len(i['AdjacentNode'])
    average_degree = sum_degree*2/float(len(adjacent_table))

    return average_degree


def cal_degree_distribution(graph):
    '''
    计算网络的度分布,返回一个字典
    '''
    degree_distribution_dict = {}
    for i in graph:
        degree_distribution_dict[i['Vertex']['Id']] = len(i['AdjacentNode'])

    return degree_distribution_dict
```

```python
def get_type(graph):
    '''
    根据存储的图的结构，得到每个节点的类型，返回一个字典
    '''
    type_dict = {}
    for i in graph:
        type_dict[i['Vertex']['Id']] = i['Vertex']['Type']

    return type_dict
```

__init__.py

```python
if __name__ == 'main':
    print("作为主程序运行")
else:
    print("NetworkBulider初始化")
```

Visualization

plotgraph.py

```python
import networkx as nx
import matplotlib.pyplot as plt

def plotdegree_distribution(lines, x_1, x_2):
    '''
    度在[x_1,x_2]范围内的的分布图
    输入节点文件即可

    因为不太需要节点的其他属性，
    只需要节点信息和边的信息，所以采用networkx库进行分析和可视化
    '''

    G = nx.Graph()
    edge_index = lines.index("*Edges\n")

    for i in lines[1:edge_index]:
        lis = i.split('\t')
        G.add_node(int(lis[0]))
    for i in lines[edge_index+1:]:
        lis = i.split('\t')
        G.add_edge(int(lis[0]), int(lis[1]), weight=1)
    print(nx.info(G))

    degree = nx.degree_histogram(G)  # 返回图中所有节点的度分布序列

    plt.bar([i for i in range(x_1, x_2+1)], degree[x_1:x_2+1],
            width=0.80, color='steelblue', alpha=0.9)

    plt.title("Frequency Histogram")
    plt.ylabel("Quantity")
    plt.xlabel("Degree")
    plt.show()


def draw_graph(lines):
```

```
    '''
    绘制节点文件的网络图

    因为不太需要节点的其他属性，
    只需要节点信息和边的信息，所以采用networkx库进行分析和可视化
    '''
    G = nx.Graph()
    edge_index = lines.index("*Edges\n")

    for i in lines[1:edge_index]:
        lis = i.split('\t')
        G.add_node(int(lis[0]))
    for i in lines[edge_index+1:]:

        lis = i.split('\t')
        G.add_edge(int(lis[0]), int(lis[1]), weight=1)

    # 输出无向图的信息
    # print("Info of the graph:\n",nx.info(G))
    # 绘制网络图
    nx.draw_shell(G,with_labels = True)
    # 展示图片
    plt.show()
```

plotnodes.py

```
import matplotlib.pyplot as plt
import collections

def plot_nodes_attr(lines, feature='type'):
    '''
    绘制图中节点属性的统计结果
    目前只有节点的type的统计绘图功能
    '''

    index_egde = lines.index("*Edges\n")

    Graph_List = []

    for line in lines[1:index_egde]:
        dic = {}
        vector = {}
        lis = line.split('\t')

        dic['Id'] = lis[0]
        dic['Name'] = lis[1]
        dic['Weight'] = lis[2]  # 因为权重信息并不用得上，所以加入字典中并不影响
        dic['Type'] = lis[3]
        dic['OtherInfo'] = lis[4].split('\n')[0].split(';')[:-1]

        vector["Vertex"] = dic
        vector["AdjacentNode"] = []
        Graph_List.append(vector)

    for line in lines[index_egde+1:]:
        lis = line.split('\t')
```

```
            if int(lis[1]) not in Graph_List[int(lis[0])]["AdjacentNode"]:
                Graph_List[int(lis[0])]["AdjacentNode"].append(int(lis[1]))
            else:
                pass

    if feature == 'type':
        type_list = []
        for i in Graph_List:
            type_list.append(i['Vertex']['Type'])

        type_dict = collections.Counter(type_list)

        plt.xlabel("Type")
        plt.ylabel("Number")
        plt.bar([1, 2, 3, 4], list(type_dict.values()),
                align='center', color=['#4E598C', '#F9C784', '#FCAF58',
'#FF8C42'], width=0.5, alpha=0.7)
        plt.ylim(ymin=0, ymax=22000)
        plt.xticks([1, 2, 3, 4], list(type_dict.keys()))
        plt.title("Node Type Distribution")
        plt.show()
    else:
        print("The feature doesn't fit.\n")
```

__init__.py

```
if __name__ == 'main':
    print("作为主程序运行")
else:
    print("Visualization初始化")
```

实现思路

数据清洗

因为原始的文件包含重复的边，而在简单的无向图中并不允许重复边的存在，所以编写data_clean()函数进行数据清洗，去掉重复边，将结果重新写入名为newnew_movies.txt的文件

数据处理

按行读入数据后，存储为字典，每个字典对应一个的节点，再将字典存储为列表。其中用EdgeTable存储某个结点的部分相邻节点信息

输出则利用format函数进行格式化输出，将某个结点的所有属性进行比较整齐美观的打印

图的序列化处理

利用pickle包将运行结果进行序列化存储，保存在名为graph_info.json的文件中

而需要再次使用时，则可以直接使用load方法进行加载

统计处理

在此处则利用之前生成的节点列表中包含的字典，采用for循环的方式进行遍历，并进行统计分析，求出图中节点的平均度

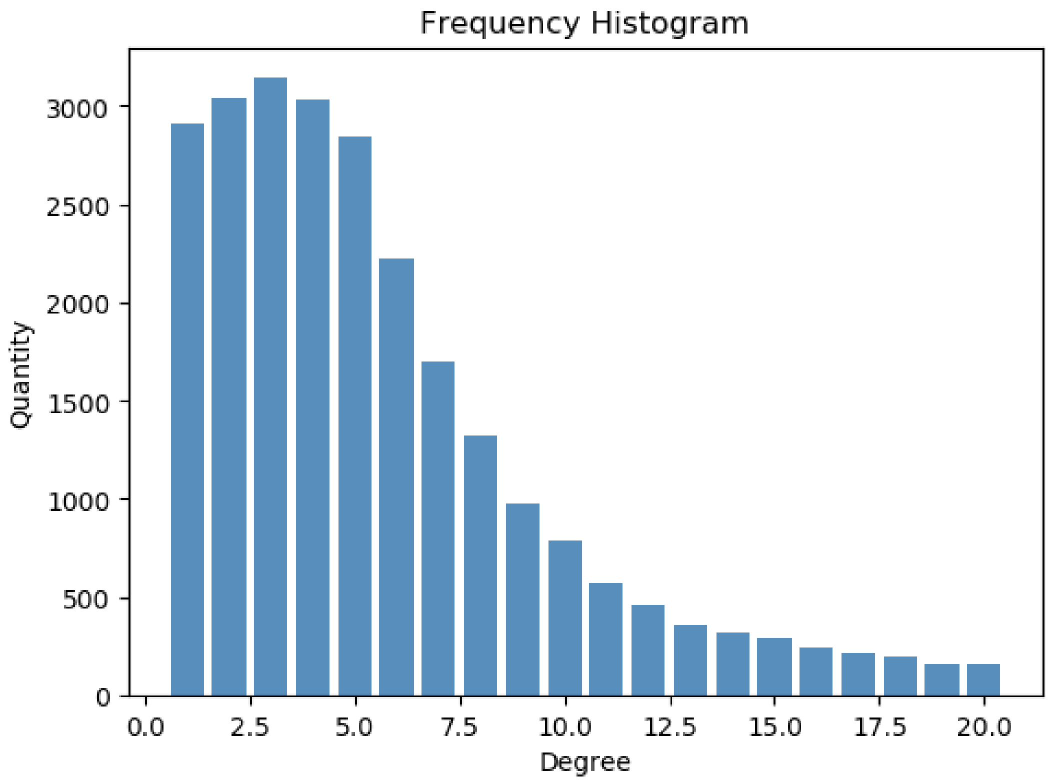
可视化

结点的度分布图

将度的范围在1-20范围内的节点进行归纳统计，绘制直方图

可以看出大部分结点的度落在[1-10]这个区间内，度越大，对应节点的比例越小

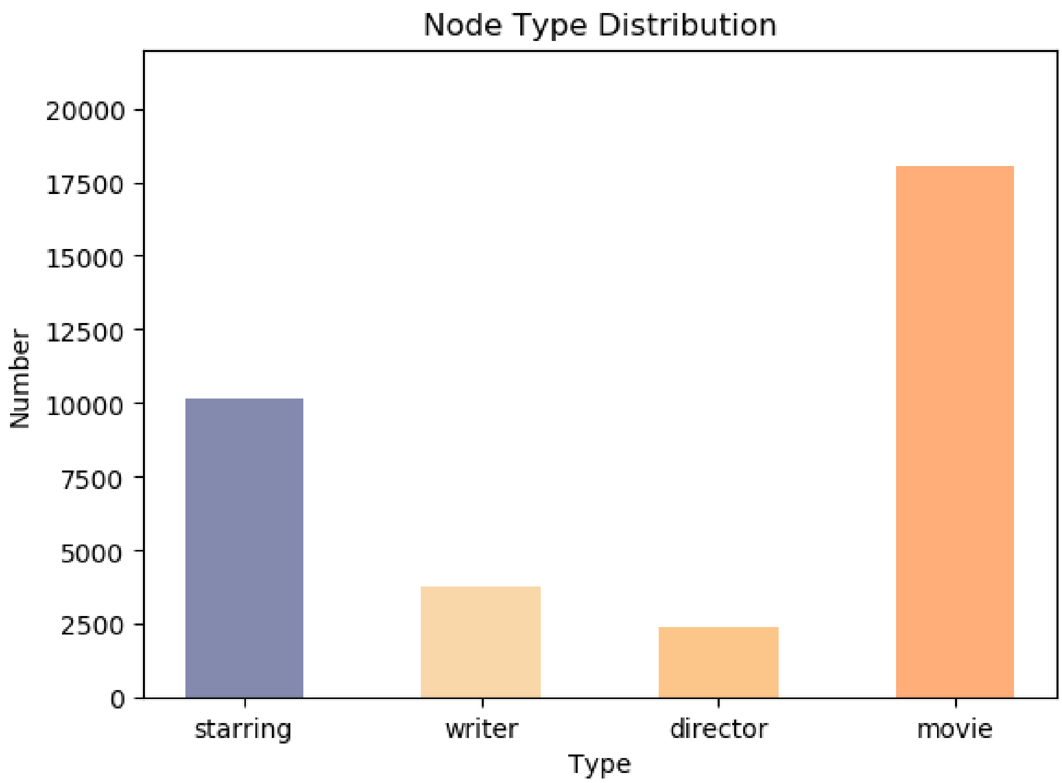
同时在GraphStat.Visualization.plotgraph模块的plotdegree_distribution函数中，可以通过调整函数的参数来进行绘图范围的控制，在此我选择了1~20这个局部的区间。区间范围越大图像会显得比较不直观，因为不同度的节点数频数相差较大。



节点类型分布

对结点的Type属性进行分析，其主要有四种表现形式，所以通过频数直方图观察每种属性对应的节点数。

可以看出在该网络中，type为movie的节点所占的比例是最大的；而director类型所占的比例最小。

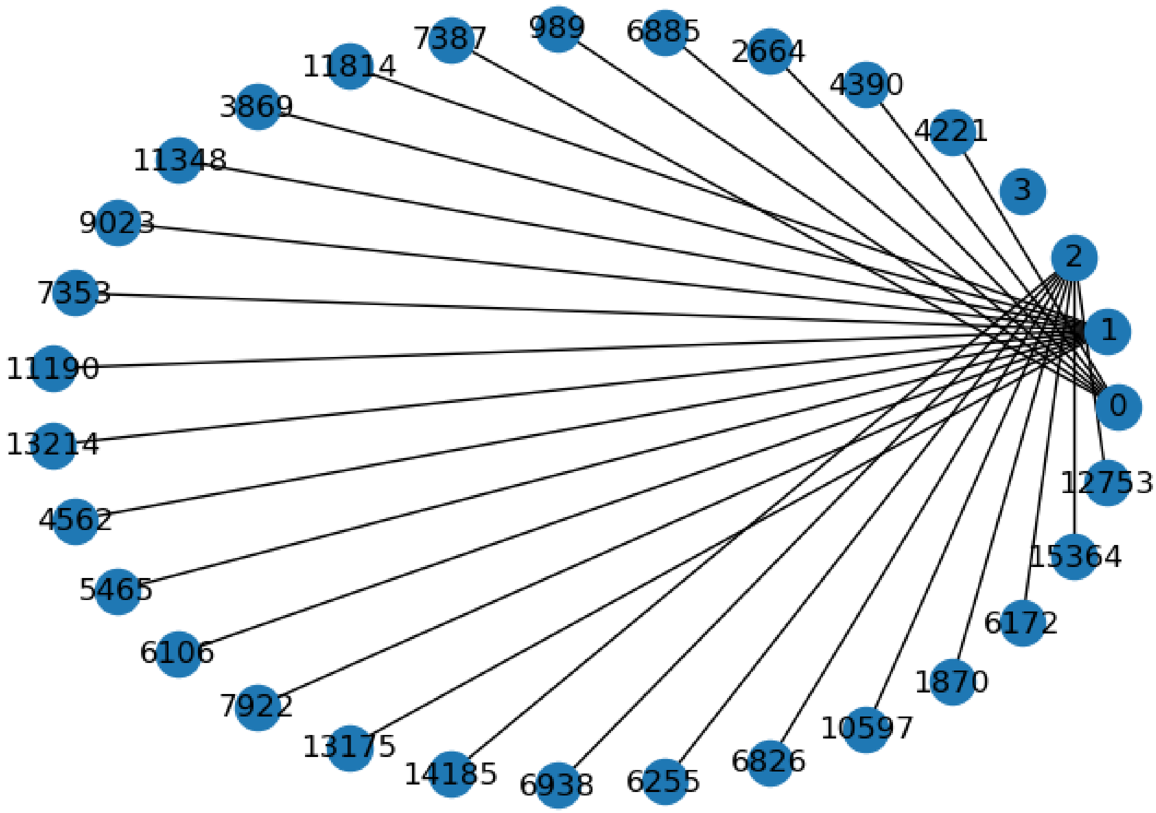


Networkx库绘制网络图

在尝试用Networkx库进行网络图形的绘制时，我首先用部分数据绘制了一个比较简单的无向图（利用nerworkx.draw_shel()函数）

但是在正式进行绘图时，我遇到了两个问题。首先是对于这个节点数和变数都非常多的无向网络，程序的运行时间将会让人难以接收。

其次，在一张图形上生成30000+个点和100000+条边，即使最后运行成功，想必也是十分的难以观察，所以我没有真正生成该网络的图形，只是对绘制网络图形的方法进行了一个大致的了解。



部分运行结果

```
*Node Infomation*
ID:                     1
Name:                   "Karen Allen"
Weight:                 7467
Node Type:              starring

*Other Information*:
American film actors
American stage actors
American video game actors
Bard College at Simon's Rock faculty
Illinois actors
People from Greene County, Illinois
Saturn Award winners

Edges Table:
[[1, 11814, 1], [1, 3869, 1], [1, 11348, 1], [1, 9023, 1], [1, 7353, 1], [1, 11190, 1], [1, 13214, 1], [1, 4562, 1], [1, 5465, 1], [1, 6106, 1], [1, 7922, 1], [1, 13175, 1], [1, 582, 1], [1, 11257, 1], [1, 16569, 1], [1, 19349, 1], [1, 23127, 1], [1, 23255, 1], [1, 25265, 1], [1, 29127, 1], [1, 31026, 1]]
The degree of the vertex is 21.

The Adjacent node of the 1 node is:
11814
3869
11348
9023
7353
11190
13214
4562
5465
6106
7922
13175
582
11257
16569
19349
23127
23255
25265
29127
31026

The average degree of the graph is:7.16.

Name:
Type: Graph
Number of nodes: 34283
Number of edges: 122706
Average degree:   7.1584
```

参考资料

https://blog.csdn.net/humanking7/article/details/88368950 Python自定义包
https://www.cnblogs.com/chengxuyuanaa/p/12079851.html Python自定义包
https://blog.csdn.net/moelimoe/article/details/105624996 Python构建无向图
http://www.zzvips.com/article/145287.html 根据邻接矩阵绘制图
https://blog.csdn.net/your_answer/article/details/79189660 networkx库使用介绍
https://networkx.github.io/documentation/networkx-1.9 Networkx库github介绍
https://www.cnblogs.com/ljhdo/p/10662902.html Networkx库使用介绍
https://www.runoob.com/python/python-modules.html python自定义包
https://www.runoob.com/python/python-modules.html Python工程的组织结构：包、模块、类
https://zhuanlan.zhihu.com/p/335347908 Python工程组织结构
https://blog.csdn.net/Qwertyuiop2016/article/details/107100345 Pickles介绍
https://cloud.tencent.com/developer/ask/62144 图的序列化存储
https://www.jb51.net/article/214863.htm Python图的存储结构
https://blog.csdn.net/weixin_38171245/article/details/99655120 Python图的存储方式
https://www.cnblogs.com/51python/p/10534237.html 利用pickle实现图的序列化存储
https://www.cnblogs.com/coolalan/p/4090740.html 度分布的介绍
https://www.cnblogs.com/qinxiaoqin/p/7150897.html Networkx生成无向图
https://my.oschina.net/u/4330791/blog/3376030 绘制度分布直方图
https://www.cnblogs.com/linblogs/p/9642472.html matplotlib学习笔记
https://www.pianshen.com/article/3252277100/ 学习python库matplotlib之常见统计柱状图(4)

相关文件

本次作业相关的代码和文件已经上传到北航云盘和个人Github代码库