

# 现代程序设计第4周作业

谢奕飞 20377077

## 程序包结构

```
1 GraphStat/
2     NetworkBuilder/
3         __init__.py
4         node.py
5             def init_node(info_path,edge_path)
6             def get_views(info,node)
7             def get_mature(info,node)
8             def get_life_time(info,node)
9             def get_created_time(info,node)
10            def get_updated_time(info,node)
11            def get_dead_account(info,node)
12            def get_language(info,node)
13            def get_link(info,node)
14            def print_mode(info,node)
15        stat.py
16            def get_node_number(node_info)
17            def get_edge_number(node_info)
18            def cal_average_dgree(node_info)
19            def cal_dgree_distribution(node_info)
20            def cal_attr_distribution(node_info,attr)
21            def load(file_path)
22        graph.py
23            def init_graph(node_info)
24            def save(data,file_name)
25            def load(file_path)
26            def init_node(info_path,edge_path)
27    Visualization/
28        plotgraph.py
29            def plot_ego(G,node)
30            def cal_dgree_distribution(node_info)
31            def plotdgree_distribution(node)
32            def load(file_path)
33        plotnode.py
34            def cal_attr_distribution(node_info,attr)
35            def plot_nodes_attr(node_info,attr)
36            def load(file_path)
```

node.py

```

1 import pandas as pd
2 from tqdm import tqdm
3
4 def init_node(info_path,edge_path):
5     '''
6     从数据文件中加载所有节点及其属性
7     返回字典,key 为节点的 ID,值为该节点对应的各属性值(字典)
8     '''
9     df_info=pd.read_csv(info_path,encoding='utf8')
10    node_info=df_info.to_dict(orient='index')
11    df_edges=pd.read_csv(edge_path,encoding='utf8')
12    for index,edge in tqdm(df_edges.iterrows(),desc='loading edges...'):
13        edge1=int(edge['numeric_id_1'])
14        edge2=int(edge['numeric_id_2'])
15        if 'link' not in node_info[edge1]:
16            node_info[edge1].update({'link':{edge2,}})
17        else:
18            node_info[edge1]['link'].add(edge2)
19        if 'link' not in node_info[edge2]:
20            node_info[edge2].update({'link':{edge1,}})
21        else:
22            node_info[edge2]['link'].add(edge1)
23    return node_info
24
25 def get_views(info,node):
26     '''
27     获取节点 node 属性
28     '''
29     return info[node]['views']
30
31 def get_mature(info,node):
32     '''
33     获取节点 node 属性
34     '''
35     return info[node]['mature']
36
37 def get_life_time(info,node):
38     '''
39     获取节点 node 属性
40     '''
41     return info[node]['life_time']
42
43 def get_created_time(info,node):
44     '''
45     获取节点 node 属性
46     '''
47     return info[node]['created_at']
48
49 def get_updated_time(info,node):
50     '''
51     获取节点 node 属性

```

```

52     '''
53     return info[node]['updated_at']
54
55 def get_dead_account(info,node):
56     '''
57     获取节点 node 属性
58     '''
59     return info[node]['dead_account']
60
61 def get_language(info,node):
62     '''
63     获取节点 node 属性
64     '''
65     return info[node]['language']
66
67 def get_link(info,node):
68     '''
69     获取节点 node 属性
70     '''
71     if 'link' in info[node]:
72         return info[node]['link']
73     else:
74         return {'NONE',}
75
76 def print_mode(info,node):
77     '''
78     显示节点全部信息 (利用 format 或者 f 函数)
79     '''
80     print('-----')
81     if 'link' in info[node]:
82         print('numeric_id: {}\nviews: {}\nmature: {}\nlife_time: {}\ncreated')
83     else:
84         print('numeric_id: {}\nviews: {}\nmature: {}\nlife_time: {}\ncreated')
85     print('-----')
86
87 if __name__=='__main__':main()

```

## init\_node()

```

1 def main():
2     '''
3     测试函数
4     '''
5     info_path='D:\Project\Python\week4module\large_twitvh_features.csv'
6     edge_path='D:\Project\Python\week4module\large_twitvh_edges.csv'
7     info=init_node(info_path,edge_path)
8     print(info)

```

```
d:\Project\Python - VS Code 控制台
loading edges...: 6797557it [11:27, 9884.46it/s]
{0: {'views': 7879, 'mature': 1, 'life_time': 969, 'created_at': '2016-02-16', 'updated_at': '2018-10-12', 'numeric_id': 0, 'dead_account': 0, 'language': 'EN', 'affiliate': 1, 'link': {'158848, 126592, 125701, 160906, 38806, 68505, 67226, 108187, 123807, 94370, 46243, 136870, 134950, 135216, 151601, 59443, 22333, 82495, 26816, 148674, 73029, 56134, 10441, 164047, 147279, 57167, 53471, 141664, 144736, 10464, 96866, 160358, 76651, 106347, 116589, 89326, 104943, 148844, 134642, 146421, 121846, 13048, 101756}}, 1: {'views': 500, 'mature': 0, 'life_time': 2699, 'created_at': '2011-05-19', 'updated_at': '2018-10-08', 'numeric_id': 1, 'dead_account': 0, 'language': 'EN', 'affiliate': 0, 'link': {'2566, 105992, 23562, 114619, 17420, 28686, 149006, 157195, 78354, 11288, 90650, 121882, 113692, 76829, 35358, 78367, 56352, 112156, 53788, 28703, 52262, 126504, 148010, 40490, 10287, 48695, 113720, 1081, 9275, 30268, 134205, 167998, 139843, 49222, 84038, 164425, 2122, 80459, 149069, 35408, 30288, 32338, 161362, 97874, 48214, 41048, 19547, 64605, 102493, 132706, 92773, 4712, 6250, 25709, 167533, 130673, 107123, 12916, 31352, 18554, 100476, 58493, 142087, 10367, 149635, 66693, 93318, 60551, 14470, 145545, 155786, 118410, 35468, 59532, 103558, 38031, 155281, 83095, 32920, 152, 23195, 107167, 5280, 61089, 60065, 122021, 40101, 28327, 54953, 95914, 137899, 104106, 22189, 124589, 2220, 142512, 130232, 139448, 57532, 42688, 131265, 158403, 123076, 85701, 26309, 53444, 125642, 93390, 77518, 54993, 133329, 140500, 134873, 8410, 74457, 113885, 46302, 62688, 57569, 163040, 87271, 152296, 119017, 15080, 83691, 16108, 146669, 8430, 9455, 112881, 128241, 118516, 86266, 119034, 85756, 106236, 85758, 2815, 117505, 84226, 144643, 149250, 67333, 115974, 65287, 92424, 113417, 110345, 108809, 111372, 90374, 127974, 19719, 87817, 57101, 48398, 94483, 127763, 104720, 87318, 131354, 6938, 59163, 90394, 104735, 107808, 131361, 35618, 61730, 54063, 149295, 86321, 144687, 126773, 14649, 146235, 1852, 37181, 34625, 134979, 75077, 102726, 31559, 128328, 27976, 152394, 107854, 154446, 77138, 152402, 61780, 11604, 23378, 146782, 6494, 128864, 43359, 108906, 99179, 131436, 21357, 104302, 78188, 99696, 117105, 89970, 85877, 37752, 161658, 73083, 41855, 98175, 103295, 87938, 5507, 94598, 68999, 13704, 71050, 8079, 63889, 147858, 32147, 104340, 55192, 73625, 32154, 922, 146335, 74656, 64419, 31140, 40868, 61862, 53670, 143272, 148905, 60842, 165799, 55207, 96173, 54701, 62382, 123824, 37297, 150964, 8121, 37818, 13754, 29118, 56255, 88518, 45001, 165834, 84427, 39372, 140745, 7632, 100305, 6097, 153554, 113620, 53724, 55261, 52703, 79844, 43498, 3051, 1515, 89069, 153070, 68588, 8176, 165361, 120819, 155127, 73213}}, 2: {'views': 382502, 'mature': 1, 'life_time': 3149, 'created_at': '2010-02-27', 'updated_at': '2018-10-12', 'numeric_id': 2, 'dead_account': 0, 'language': 'EN', 'affiliate': 1, 'link': {'125959, 48141, 37906, 129054, 113183, 103455, 56358, 138791, 92203, 156214, 87610, 15936, 96322, 87619, 74819, 31815, 67143, 52298, 131146, 87631, 99921, 96850, 21587, 114770, 124509, 71262, 104541, 141418, 121450, 77421, 86637, 12399, 97396, 38527, 101507, 50313, 84618, 80010, 101527, 51871, 98475, 88751, 123059, 77495, 48316, 100542, 42177, 98503, 101576, 75467, 83660, 73944, 153307, 98524, 29419, 17646, 100593, 86769, 32506, 3323, 78592, 99073, 37637, 70919, 134927, 80656, 14608, 106258, 114450, 127765, 10520, 5404, 50978, 63266, 93477, 99626, 136508, 19773, 53054, ...
}
```

功能：将csv文件中的节点信息和边信息转化为嵌套字典

## print\_mode()

```
1 def main():
2     '''
3     测试函数
4     '''
5     info_path='D:\Project\Python\week4module\large_twitch_features.csv'
6     edge_path='D:\Project\Python\week4module\large_twitch_edges1.csv'
7     info=init_node(info_path,edge_path)
8     print_mode(info,200)
9     print_mode(info,300)
10    print_mode(info,400)
11    print_mode(info,500)
12    print_mode(info,700)
```

```
d:\Project\Python - VS Code 控制台
loading edges...: 999it [00:00, 9641.30it/s]

-----
numeric_id: 200
views: 14486
mature: 1
life_time: 1896
created_at: 2013-08-03
updated_at: 2018-10-12
dead_account: 0
language: ZH
link: NONE
-----
numeric_id: 300
views: 3396
mature: 1
life_time: 1751
created_at: 2013-12-22
updated_at: 2018-10-08
dead_account: 0
language: EN
link: NONE
-----
numeric_id: 400
views: 18620
mature: 0
life_time: 1561
created_at: 2014-07-04
updated_at: 2018-10-12
dead_account: 0
language: EN
link: NONE
-----
numeric_id: 500
views: 936
mature: 0
life_time: 2094
created_at: 2012-12-11
updated_at: 2018-09-05
dead_account: 0
language: EN
link: NONE
-----
numeric_id: 700
views: 981
mature: 0
life_time: 2207
created_at: 2012-09-25
updated_at: 2018-10-11
dead_account: 0
language: ES
link: [141493]
-----
请按任意键继续. . .
```

(以large\_twitch\_edges.csv前1000行数据为例)

## get\_XXX()

```
1 def main():
2     '''
3     测试函数
4     '''
5     info_path='D:\Project\Python\week4module\large_twitch_features.csv'
6     edge_path='D:\Project\Python\week4module\large_twitch_edges1.csv'
7     ID=200
8     info=init_node(info_path,edge_path)
9     print('numeric_id: ',ID)
10    print('views: ',get_views(info,ID))
11    print('mature: ',get_mature(info,ID))
12    print('life_time: ',get_life_time(info,ID))
13    print('created_at: ',get_created_time(info,ID))
14    print('updated_at: ',get_updated_time(info,ID))
15    print('dead_account: ',get_dead_account(info,ID))
16    print('language: ',get_language(info,ID))
17    print('link: ',get_link(info,ID))
18    print()
19    ID=700
20    print('numeric_id: ',ID)
```

```

21     print('views: ',get_views(info,ID))
22     print('mature: ',get_mature(info,ID))
23     print('life_time: ',get_life_time(info,ID))
24     print('created_at: ',get_created_time(info,ID))
25     print('updated_at: ',get_updated_time(info,ID))
26     print('dead_account: ',get_dead_account(info,ID))
27     print('language: ',get_language(info,ID))
28     print('link: ',get_link(info,ID))

```

```

d:\Project\Python - VS Code 控制台
loading edges...: 999it [00:00, 9602.46it/s]
numeric_id: 200
views: 14486
mature: 1
life_time: 1896
created_at: 2013-08-03
updated_at: 2018-10-12
dead_account: 0
language: ZH
link: {'NONE'}

numeric_id: 700
views: 981
mature: 0
life time: 2207
created_at: 2012-09-25
updated_at: 2018-10-11
dead_account: 0
language: ES
link: {141493}
请按任意键继续...

```

(以large\_twitch\_edges.csv前1000行数据为例)

功能：获得某个节点的XXX属性

## graph.py

```

1  import pandas as pd
2  from tqdm import tqdm
3  import pickle
4  import os
5  import sys
6  import networkx as nx
7
8  def init_graph(node_info):
9      '''
10     构建网络
11     '''
12     G=nx.Graph()
13     for each in tqdm(node_info,desc='loading node_info...'):
14         if 'link' not in node_info[each]:
15             continue
16         for link in node_info[each]['link']:
17             G.add_edge(each,link)
18     return G
19
20 def save(data,file_name):
21     '''
22     序列化信息
23     '''
24     file_path=str(sys.argv[0])[:-len(os.path.basename(sys.argv[0]))]+file_na

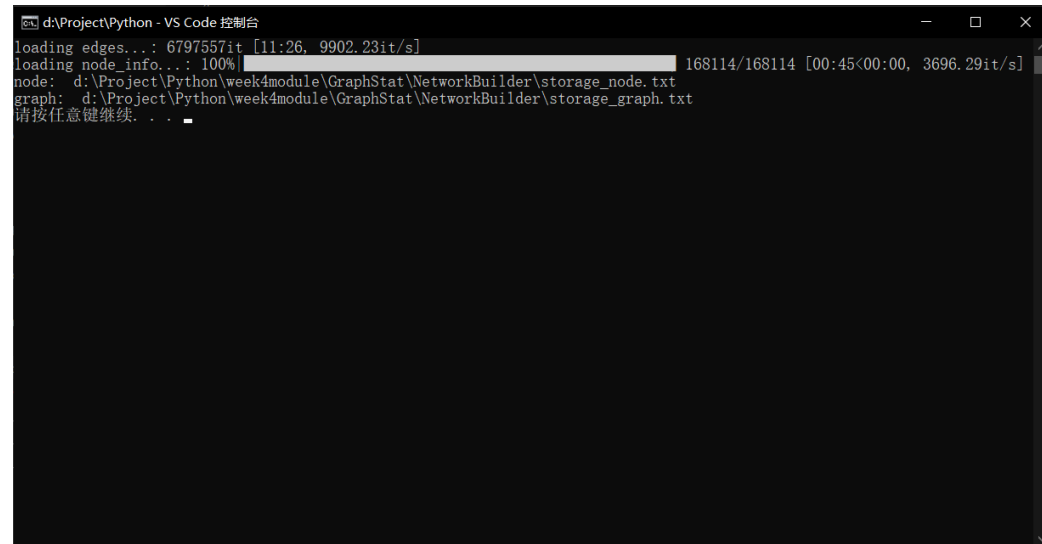
```

```

25     f=open(file_path,'wb')
26     pickle.dump(data,f,0)
27     f.close
28     return file_path
29
30 def load(file_path):
31     '''
32     将序列化信息加载至内存
33     '''
34     f=open(file_path,'rb')
35     data=pickle.load(f)
36     f.close()
37     print(file_path,' loaded √')
38     return data
39
40 def init_node(info_path,edge_path):
41     '''
42     从数据文件中加载所有节点及其属性
43     返回字典,key 为节点的 ID,值为该节点对应的各属性值(字典)
44     '''
45     df_info=pd.read_csv(info_path,encoding='utf8')
46     node_info=df_info.to_dict(orient='index')
47     df_edges=pd.read_csv(edge_path,encoding='utf8')
48     for index,edge in tqdm(df_edges.iterrows(),desc='loading edges...'):
49         edge1=int(edge['numeric_id_1'])
50         edge2=int(edge['numeric_id_2'])
51         if 'link' not in node_info[edge1]:
52             node_info[edge1].update({'link':{edge2,}})
53         else:
54             node_info[edge1]['link'].add(edge2)
55         if 'link' not in node_info[edge2]:
56             node_info[edge2].update({'link':{edge1,}})
57         else:
58             node_info[edge2]['link'].add(edge1)
59     return node_info
60
61 def main():
62     '''
63     测试函数
64     '''
65     node=load('d:/Project/Python/week4module/GraphStat/NetworkBuilder/storag
66     graph=load('D:/Project/Python/week4module/GraphStat/NetworkBuilder/stora
67     print()
68

```

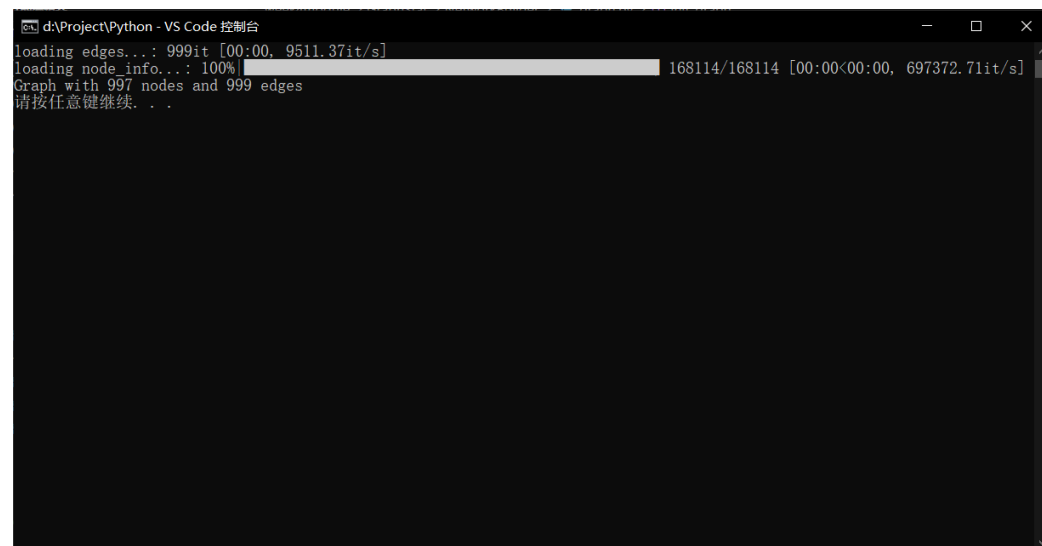
```
if __name__=='__main__': main()if __name__=='__main__': main()
```



```
d:\Project\Python - VS Code 控制台
loading edges...: 6797557it [11:26, 9902.23it/s]
loading node info...: 100% 168114/168114 [00:45<00:00, 3696.29it/s]
node: d:\Project\Python\week4module\GraphStat\NetworkBuilder\storage_node.txt
graph: d:\Project\Python\week4module\GraphStat\NetworkBuilder\storage_graph.txt
请按任意键继续. . .
```

## init\_graph()

```
1 def main():
2     '''
3     测试函数
4     '''
5     info_path='D:\Project\Python\week4module\large_twitch_features.csv'
6     edge_path='D:\Project\Python\week4module\large_twitch_edges1.csv'
7     node_info=init_node(info_path,edge_path)
8     graph=init_graph(node_info)
9     print(graph)
```



```
d:\Project\Python - VS Code 控制台
loading edges...: 999it [00:00, 9511.37it/s]
loading node info...: 100% 168114/168114 [00:00<00:00, 697372.71it/s]
Graph with 997 nodes and 999 edges
请按任意键继续. . .
```

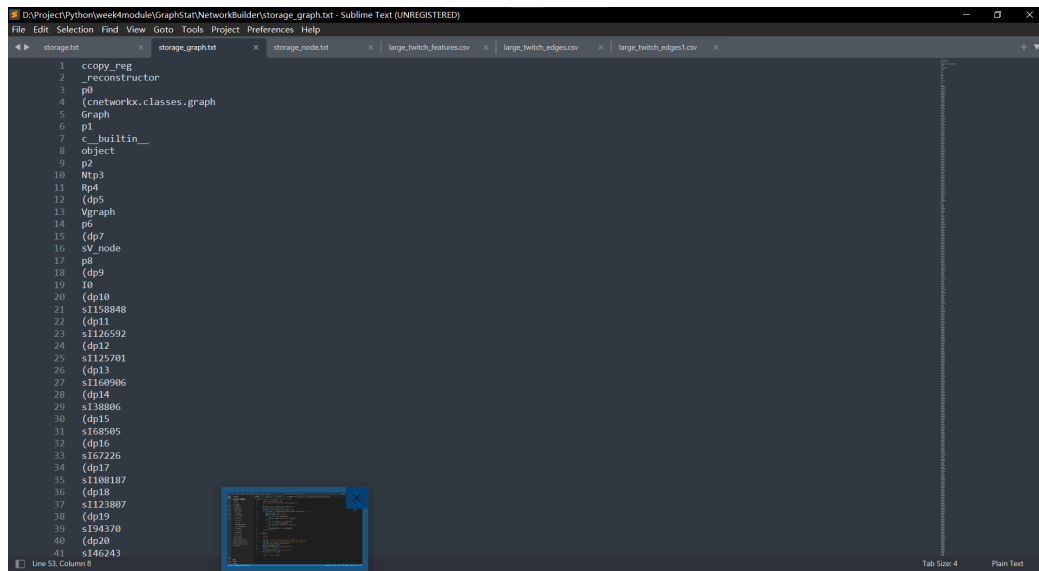
(以large\_twitch\_edges.csv前1000行数据为例)

**功能：**从node\_info的link中读取边的信息，构建节点与所连节点的字典

## save()

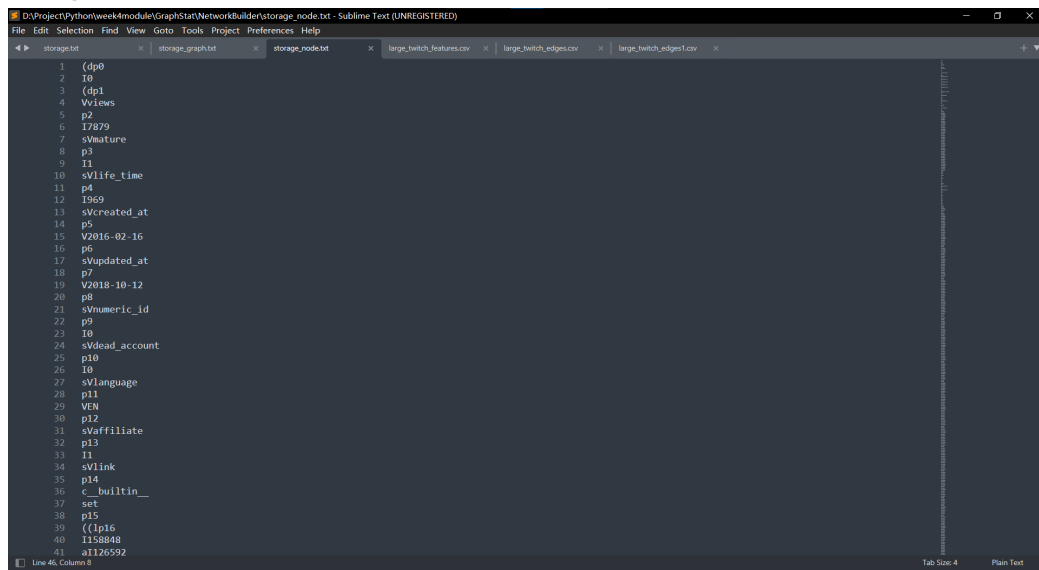
storage\_graph.txt





```
1 ccopy_reg
2 reconstructor
3 p0
4 (cnetworkx.classes.graph
5 Graph
6 p1
7 c_builtin_
8 object
9 p2
10 Mtp3
11 Rp4
12 (dp5
13 Vgraph
14 p6
15 (dp7
16 sv_node
17 p8
18 (dp9
19 10
20 (dp10
21 s1158848
22 (dp11
23 s1126592
24 (dp12
25 s1125701
26 (dp13
27 s1160906
28 (dp14
29 s138806
30 (dp15
31 s165505
32 (dp16
33 s167226
34 (dp17
35 s1168187
36 (dp18
37 s1123807
38 (dp19
39 s194370
40 (dp20
41 s146243
```

## storage\_node.txt



```
1 (dp0
2 10
3 (dp1
4 Vviews
5 p2
6 17879
7 sVnature
8 p3
9 11
10 sVlife_time
11 p4
12 1969
13 sVcreated_at
14 p5
15 V2016-02-16
16 p6
17 sVupdated_at
18 p7
19 V2018-10-12
20 p8
21 sVnumeric_id
22 p9
23 10
24 sVdead_account
25 p10
26 10
27 sVlanguage
28 p11
29 VEN
30 p12
31 sVaffiliate
32 p13
33 11
34 sVlink
35 p14
36 c_builtin_
37 set
38 p15
39 (11615
40 1158848
41 a1126592
```

功能：将信息序列化存储在当前文件夹下的自定义文件中

## load()

```
1 def main():
2     '''
3     测试函数
4     '''
5     node=load('d:/Project/Python/week4module/GraphStat/NetworkBuilder/storag
6     print(node)
```

```
d:\Project\Python - VS Code 控制台
[0: {'views': 7879, 'mature': 1, 'life_time': 969, 'created_at': '2016-02-16', 'updated_at': '2018-10-12', 'numeric_id': 0, 'dead_account': 0, 'language': 'EN', 'affiliate': 1, 'link': {158848, 126592, 125701, 160906, 38806, 68505, 67226, 108187, 123807, 94370, 46243, 136870, 134950, 135216, 151601, 59443, 22333, 82495, 26816, 148674, 73029, 56134, 10441, 164047, 147279, 57167, 53471, 141664, 144736, 10464, 96866, 160358, 76651, 106347, 116589, 89326, 104943, 148844, 134642, 146421, 121846, 13048, 101756}}, 1: {'views': 500, 'mature': 0, 'life_time': 2699, 'created_at': '2011-05-19', 'updated_at': '2018-10-08', 'numeric_id': 1, 'dead_account': 0, 'language': 'EN', 'affiliate': 0, 'link': {2566, 105992, 23562, 14619, 17420, 28686, 149006, 157195, 78354, 11288, 121882, 90650, 112156, 113692, 76829, 35358, 56352, 78367, 53788, 28703, 52262, 126504, 148010, 40490, 10287, 48695, 113720, 1081, 9275, 30268, 134205, 167998, 139843, 49222, 84038, 16442, 5, 2122, 80459, 149069, 35408, 30288, 32338, 161362, 97874, 48214, 41048, 19547, 64605, 102493, 132706, 92773, 4712, 6250, 25709, 167533, 130673, 107123, 12916, 31352, 18554, 100476, 58493, 10367, 149635, 66693, 93318, 60551, 103558, 14470, 155786, 118410, 35468, 59532, 145545, 38031, 155281, 83095, 32920, 152, 23195, 107167, 5280, 61089, 60065, 122021, 4010, 1, 28327, 54953, 95914, 137899, 104106, 22189, 124589, 2220, 142512, 130232, 139448, 57532, 42688, 131265, 158403, 12307, 6, 85701, 26309, 53444, 125642, 93390, 77518, 54993, 133329, 140500, 134873, 8410, 74457, 113885, 46302, 62688, 57569, 163040, 87271, 152296, 119017, 15080, 83691, 16108, 146669, 8430, 9455, 112881, 128241, 118516, 86266, 119034, 85756, 106236, 85758, 2815, 117505, 84226, 144643, 149250, 67333, 115974, 142087, 65287, 92424, 113417, 110345, 108809, 111372, 90374, 72974, 19719, 87817, 57101, 48398, 94483, 127763, 104720, 87318, 131354, 6938, 59163, 90394, 104735, 107808, 131361, 35618, 61730, 54063, 149295, 86321, 144687, 126773, 14649, 146235, 1852, 37181, 34625, 134979, 75077, 102726, 31559, 128328, 27976, 152394, 107854, 154446, 77138, 152402, 61780, 11604, 23378, 146782, 6494, 128864, 43359, 108906, 99179, 131436, 21357, 104302, 78188, 99696, 117105, 89970, 85877, 37752, 161658, 73083, 41855, 98175, 103295, 87938, 5507, 94598, 68999, 13704, 71050, 8079, 63889, 147858, 32147, 104340, 55192, 73625, 32154, 922, 146335, 74656, 64419, 31140, 40868, 61862, 53670, 143272, 148905, 60842, 165799, 55207, 96173, 54701, 62382, 123824, 37297, 150964, 8121, 37818, 13754, 29118, 56255, 88518, 45001, 165834, 84427, 39372, 140745, 7632, 100305, 6097, 153554, 113620, 53724, 55261, 52703, 79844, 43498, 3051, 1515, 89069, 153070, 68588, 8176, 165361, 120819, 155127, 73213}}, 2: {'views': 382502, 'mature': 1, 'life_time': 3149, 'created_at': '2010-02-27', 'updated_at': '2018-10-12', 'numeric_id': 2, 'dead_account': 0, 'language': 'EN', 'affiliate': 1, 'link': {125959, 48141, 37906, 129054, 103455, 113183, 56358, 138791, 92203, 156214, 87610, 15936, 9632, 87619, 74819, 31815, 67143, 52298, 131146, 87631, 99921, 96850, 21587, 114770, 124509, 71262, 104541, 141418, 121450, 77421, 86637, 12399, 97396, 38527, 101507, 50313, 84618, 80010, 101527, 51871, 98475, 88751, 123059, 77495, 48316, 100542, 42177, 98503, 101576, 75467, 83660, 73944, 153307, 98524, 29419, 17646, 100593, 86769, 32506, 3323, 78592, 99073, 37637, 70919, 134927, 80656, 14608, 106258, 114450, 127765, 10520, 5404, 50978, 63266, 93477, 99626, 136508, 19773, 53054, 70463, 158527, 124745, 93001, 146254, 130385, 11094, 63321, 98143, 79202, 58210, 99176, 43891, 153468, 2428, 121730, 55...
```

```
1 def main():
2     '''
3     测试函数
4     '''
5     graph=load('D:/Project/Python/week4module/GraphStat/NetworkBuilder/storage_graph.txt')
6     print(graph)
```

```
d:\Project\Python - VS Code 控制台
Graph with 168114 nodes and 6797557 edges
请按任意键继续. . .
```

功能：通过路径载入信息

## stat.py

```
1 from tqdm import tqdm
2 import pickle
3
4 def get_node_number(node_info):
5     '''
6     计算节点数
7     '''
8     node=set()
9     for each in tqdm(node_info,desc='getting node_number...'):
10         node.add(each)
11         if 'link' in node_info[each]:
12             node.add(link for link in node_info[each]['link'])
```

```

13     return len(node)
14
15 def get_edge_number(node_info):
16     '''
17     计算边数
18     '''
19     edge=set()
20     for each in tqdm(node_info,desc='getting edge_number...'):
21         if 'link' in node_info[each]['link']:
22             for link in node_info[each]['link']:
23                 edge.add(set(each,link))
24         else:
25             continue
26     return len(edge)
27
28 def cal_average_dgree(node_info):
29     '''
30     计算网络中的平均度
31     '''
32     dgrees=0
33     node_number=0
34     for each in tqdm(node_info,desc='calculating average_dgree...'):
35         if 'link' in node_info[each]:
36             dgrees+=len(node_info[each]['link'])
37             node_number+=1
38     return dgrees/node_number
39
40 def cal_dgree_distribution(node_info):
41     '''
42     计算网络的度分布
43     返回度与频数的字典
44     '''
45     dgrees={}
46     for each in node_info:
47         if 'link' in node_info[each]:
48             dgree=len(node_info[each]['link'])
49             if dgree not in dgrees:
50                 dgrees.update({dgree:1})
51             else:
52                 dgrees[dgree]+=1
53     return dgrees
54
55 def cal_attr_distribution(node_info,attr):
56     '''
57     计算attr属性的分布
58     返回attr与频数的字典
59     '''
60     v={}
61     for each in node_info:
62         if node_info[each][attr] not in v:
63             v.update({node_info[each][attr]:1})

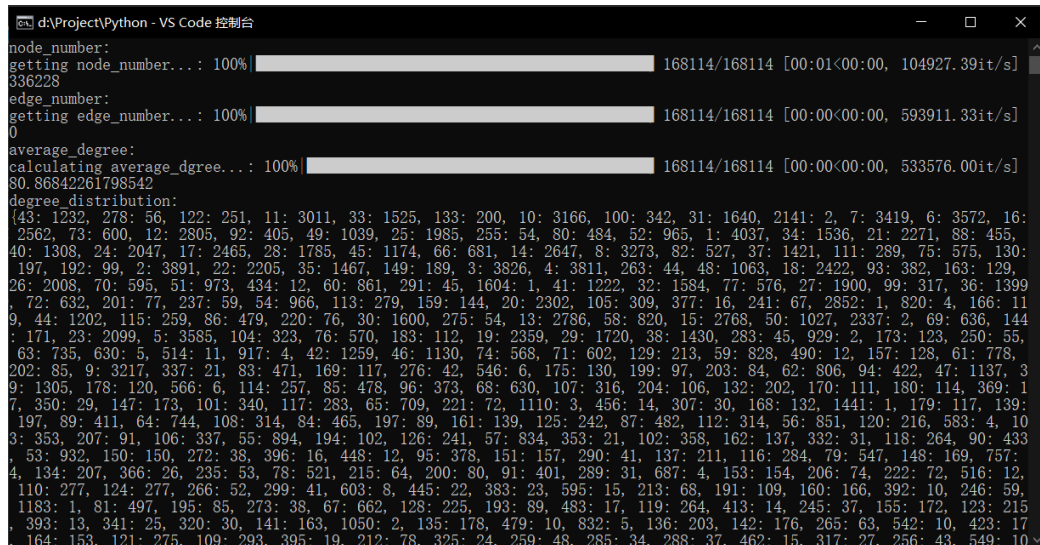
```

```

64         else:
65             v[node_info[each][attr]]+=1
66     return v
67
68 def load(file_path):
69     '''
70     将网络加载至内存
71     '''
72     f=open(file_path,'rb')
73     data=pickle.load(f)
74     f.close()
75     print(file_path,' loaded √')
76     return data
77
78 def main():
79     '''
80     测试函数
81     '''
82     node=load('d:/Project/Python/week4module/GraphStat/NetworkBuilder/storag
83     print('node_number: ')
84     print(get_node_number(node))
85     print('edge_number: ')
86     print(get_edge_number(node))
87     print('average_degree: ')
88     print(cal_average_dgree(node))
89     print('degree_distribution: ')
90     print(cal_dgree_distribution(node))
91     print('views_distribution: ')
92     print(cal_attr_distribution(node,'views'))
93
94
95 if __name__=='__main__':

```

main() main()



```

d:\Project\Python - VS Code 控制台
node_number:
getting node_number...: 100% 168114/168114 [00:01<00:00, 104927.39it/s]
336228
edge_number:
getting edge_number...: 100% 168114/168114 [00:00<00:00, 593911.33it/s]
0
average_degree:
calculating average_dgree...: 100% 168114/168114 [00:00<00:00, 533576.00it/s]
80.86842261798542
degree_distribution:
(43: 1232, 278: 56, 122: 251, 11: 3011, 33: 1525, 133: 200, 10: 3166, 100: 342, 31: 1640, 2141: 2, 7: 3419, 6: 3572, 16:
2562, 73: 600, 12: 2805, 92: 405, 49: 1039, 25: 1985, 255: 54, 80: 484, 52: 965, 1: 4037, 34: 1536, 21: 2271, 88: 455,
40: 1308, 24: 2047, 17: 2465, 28: 1785, 45: 1174, 66: 681, 14: 2647, 8: 3273, 82: 527, 37: 1421, 111: 289, 75: 575, 130:
197, 192: 99, 2: 3891, 22: 2205, 35: 1467, 149: 189, 3: 3826, 4: 3811, 263: 44, 48: 1063, 18: 2422, 93: 382, 163: 129,
26: 2008, 70: 595, 51: 973, 434: 12, 60: 861, 291: 45, 1604: 1, 41: 1222, 32: 1584, 77: 576, 27: 1900, 99: 317, 36: 1399
, 72: 632, 201: 77, 237: 59, 54: 966, 113: 279, 159: 144, 20: 2302, 105: 309, 377: 16, 241: 67, 2852: 1, 820: 4, 166: 11
9, 44: 1202, 115: 259, 86: 479, 220: 76, 30: 1600, 275: 54, 13: 2786, 58: 820, 15: 2768, 50: 1027, 2337: 2, 69: 636, 144
: 171, 23: 2099, 5: 3585, 104: 323, 76: 570, 183: 112, 19: 2359, 29: 1720, 38: 1430, 283: 45, 929: 2, 173: 123, 250: 55,
63: 735, 630: 5, 514: 11, 917: 4, 42: 1259, 46: 1130, 74: 568, 71: 602, 129: 213, 59: 828, 490: 12, 157: 128, 61: 778,
202: 85, 9: 3217, 337: 21, 83: 471, 169: 117, 276: 42, 546: 6, 175: 130, 199: 97, 203: 84, 62: 806, 94: 422, 47: 1137, 3
9: 1305, 178: 120, 566: 6, 114: 257, 85: 478, 96: 373, 68: 630, 107: 316, 204: 106, 132: 202, 170: 111, 180: 114, 369: 1
7, 350: 29, 147: 173, 101: 340, 117: 283, 65: 709, 221: 72, 1110: 3, 456: 14, 307: 30, 168: 132, 1441: 1, 179: 117, 139:
197, 89: 411, 64: 744, 108: 314, 84: 465, 197: 89, 161: 139, 125: 242, 87: 482, 112: 314, 56: 851, 120: 216, 583: 4, 10
3: 353, 207: 91, 106: 337, 55: 894, 194: 102, 126: 241, 57: 834, 353: 21, 102: 358, 162: 137, 332: 31, 118: 264, 90: 433
53: 932, 150: 150, 272: 38, 396: 16, 448: 12, 95: 378, 151: 157, 290: 41, 137: 211, 116: 284, 79: 547, 148: 169, 757:
4, 134: 207, 366: 26, 235: 53, 78: 521, 215: 64, 200: 80, 91: 401, 289: 31, 687: 4, 153: 154, 206: 74, 222: 72, 516: 12,
110: 277, 124: 277, 266: 52, 299: 41, 603: 8, 445: 22, 383: 23, 595: 15, 213: 68, 191: 109, 160: 166, 392: 10, 246: 59,
1183: 1, 81: 497, 195: 85, 273: 38, 67: 662, 128: 225, 193: 89, 483: 17, 119: 264, 413: 14, 245: 37, 155: 172, 123: 215
393: 13, 341: 25, 320: 30, 141: 163, 1050: 2, 135: 178, 479: 10, 832: 5, 136: 203, 142: 176, 265: 63, 542: 10, 423: 17
164: 153, 121: 275, 109: 293, 395: 19, 212: 78, 325: 24, 259: 48, 285: 34, 288: 37, 462: 15, 317: 27, 256: 43, 549: 10
...

```

```

d:\Project\Python - VS Code 控制台
1703: 1, 2342: 1, 1088: 1, 2213: 1, 1765: 1, 4281: 1, 1184: 1, 3255: 1, 2660: 1, 4168: 1, 955: 1, 4511: 1, 1400: 1, 295
8: 1, 18349: 1, 1698: 1, 3536: 1, 3664: 1, 8872: 1, 1023: 1, 2359: 1, 933: 1, 1188: 1, 924: 1, 3827: 1, 4938: 1, 1071: 1
, 1390: 1, 990: 1, 2979: 1, 1448: 1, 1116: 1, 4688: 1, 1317: 1, 1178: 1, 2504: 1, 2694: 1, 4843: 1, 1220: 1, 1065: 1, 23
32: 1, 9316: 1, 1480: 1, 3632: 1, 1121: 1, 2434: 1, 3777: 1, 1829: 1, 8381: 1, 1095: 1, 5239: 1, 992: 1, 1945: 1, 2608:
1, 1210: 1, 2154: 1, 1442: 1, 2131: 1, 831: 1, 1394: 1, 2531: 1, 9156: 1}
views_distribution:
{7879: 3, 500: 39, 382502: 1, 386: 38, 2486: 20, 4987: 8, 234: 30, 775: 33, 69020: 1, 32073: 1, 94: 28, 10254468: 1, 217
8: 30, 4871: 9, 2002: 27, 33882: 2, 200: 39, 11624: 5, 1160: 24, 144812: 1, 1885: 22, 52160: 1, 212680: 1, 14538: 5, 339
218: 1, 1025: 29, 9528: 1, 2802: 16, 162078: 1, 2585: 19, 1370: 33, 4405: 8, 21502: 1, 745: 30, 17042: 3, 2559: 10, 5950
: 13, 8231: 3, 879: 41, 3615: 14, 2497: 16, 19442: 2, 8541: 2, 3874: 14, 14944: 2, 46546: 1, 711: 36, 9273: 6, 1566: 26,
5667: 5, 123420: 1, 5863: 5, 370: 49, 795: 19, 9713: 4, 6043: 6, 28380: 1, 256: 29, 8788: 1, 383: 41, 856821: 1, 744: 3
0, 1168: 29, 12359: 5, 996: 26, 5594: 5, 39110: 1, 929459: 1, 9350: 2, 4594: 9, 3059: 15, 75: 29, 5639: 9, 44529: 2, 104
2: 33, 784: 31, 1958: 21, 3409: 9, 49854: 1, 33356: 1, 7738: 4, 77565: 1, 255: 27, 94298: 1, 2639: 22, 1974604: 1, 4740:
10, 9529: 2, 7149: 7, 1442: 20, 18460: 1, 3477: 10, 3955: 8, 2003: 23, 922: 33, 1108: 21, 257: 35, 47789: 1, 2154: 19,
2060: 26, 2073: 25, 164810: 1, 9429: 1, 5444: 9, 10540: 4, 103322: 1, 8436: 6, 2203: 28, 1946: 10, 10529: 4, 2112: 16, 1
19221: 1, 3945: 7, 295: 36, 8766: 9, 4093: 6, 13003233: 1, 1971793: 1, 120872: 1, 4553: 7, 140: 33, 411: 37, 306: 44, 23
07: 17, 5581: 6, 124: 26, 60918: 1, 147: 31, 162: 23, 6456: 6, 10759: 3, 2323: 21, 34130: 1, 793: 34, 2758: 13, 16797: 1
, 2479: 17, 170155: 1, 507: 31, 41660: 1, 807: 35, 4374: 8, 13180: 3, 794: 27, 11194: 2, 4279: 10, 3453: 14, 3482: 4, 21
02612: 1, 5951: 4, 204: 47, 10509729: 1, 630: 30, 43650: 2, 14557: 2, 66068: 1, 911: 26, 24020: 1, 1357: 13, 1064: 34, 2
38: 36, 1341: 25, 3968: 13, 22228: 1, 13697: 5, 9116: 5, 26427: 1, 1921: 11, 6051: 7, 313084: 1, 6144: 7, 394: 30, 578:
29, 7032: 5, 7355: 5, 832: 37, 472: 33, 11603: 5, 976: 19, 5373: 5, 1518: 23, 362: 33, 934086: 1, 785333: 1, 1448: 25, 4
30: 35, 27452: 1, 193768: 1, 6525: 6, 2850: 9, 27815: 2, 2179: 22, 11705: 3, 1326: 23, 960: 29, 19959321: 1, 27924: 1, 2
356: 20, 4597: 7, 14486: 1, 1086: 24, 5989: 11, 2863112: 1, 3929: 10, 3422: 11, 3444: 10, 12276: 6, 6701: 2, 3101: 15, 8
723: 6, 8052: 3, 5189: 12, 240: 32, 88: 31, 5735: 5, 382: 41, 7970: 7, 2871: 16, 1890: 21, 251: 34, 1081: 25, 1196: 20,
26824: 4, 15096: 4, 3521: 16, 266842: 1, 2435: 21, 1138: 25, 33004: 1, 68338: 1, 1907: 13, 1635: 32, 393: 30, 3433: 16,
1973: 21, 291260: 1, 111877: 1, 4482: 10, 702: 24, 3729: 13, 203364: 1, 59022: 1, 1058: 20, 16344: 2, 791: 26, 1034: 22,
2822: 12, 364: 32, 24703: 2, 37236: 3, 2013: 24, 25443: 1, 2333: 15, 3994: 9, 51209: 1, 46796: 2, 180: 33, 1002: 34, 17
627: 2, 8082: 5, 6370: 7, 2798: 18, 1520: 21, 565: 31, 4953: 9, 2841: 18, 44091: 1, 1596: 20, 63318: 2, 4103: 6, 4854942
: 1, 5298: 11, 583: 43, 297174: 1, 7017: 9, 5191: 3, 3340: 7, 1985: 15, 42501: 1, 10058: 3, 17344: 4, 47164: 1, 10692: 2
, 2650: 19, 2145045: 1, 1261: 28, 5421: 8, 7186: 5, 16086: 4, 3956: 11, 151991: 1, 22458: 2, 713: 25, 16322: 2, 35482: 2

```

函数	功能
get_node_number(node_info)	返回节点个数
get_edge_number(node_info)	返回边个数
cal_average_dgree(node_info)	返回平均值
cal_dgree_distribution(node_info)	返回度与频数的字典
cal_attr_distribution(node_info,attr)	返回attr与频数的字典

## plotgraph.py

```

1 import networkx as nx
2 import matplotlib.pyplot as plt
3 import pickle
4 from tqdm import tqdm
5
6 def plot_ego(G,node):
7     '''
8     （附加：使用 networkx 库中的布局算法可视化结构，
9     注意避免结构太大，复杂可能导致绘制失败，或者杂乱。）
10    绘制节点的局部网络（找一些度大小合适的节点尝试。）
11    '''
12    G_neighbor=nx.ego_graph(G,node)
13    nx.draw(G_neighbor,with_labels=True)
14    plt.show()
15    return 1
16
17 def cal_dgree_distribution(node_info):
18     '''
19     计算网络的度分布
20     返回度与频数的字典
21     '''
22     dgrees={}
23     for each in node_info:
24         if 'link' in node_info[each]:
25             dgree=len(node_info[each]['link'])

```

```

26         if dgree not in dgrees:
27             dgrees.update({dgree:1})
28         else:
29             dgrees[dgree]+=1
30     return dgrees
31
32 def plotdgree_distribution(node):
33     '''
34     ( 观察度分布的形态 )
35     度的分布图
36     '''
37     dgree=cal_dgree_distribution(node)
38     '''x=sorted(dgree.keys())
39     y=[dgree[key] for key in tqdm(x,desc='loading y...')]
40     plt.bar(x,y)
41     plt.xlabel('degree')
42     plt.ylabel('distribution')
43     plt.xlim(0,max(x))#尺度自适应
44     plt.ylim(0,max(y))
45     plt.show()
46     return 1'''
47     x=dgree.keys()
48     plt.hist(x,512)
49     plt.xlabel('degree')
50     plt.ylabel('distribution')
51     plt.show()
52     return 1
53
54 def load(file_path):
55     '''
56     将网络加载至内存
57     '''
58     f=open(file_path,'rb')
59     data=pickle.load(f)
60     f.close()
61     print(file_path,' loaded √')
62     return data
63
64 def main():
65     '''
66     测试函数
67     '''
68     node=load('d:/Project/Python/week4module/GraphStat/NetworkBuilder/storag
69     plotdgree_distribution(node)
70
71 if __name__=='__main__': main()

```

## plot\_ego()

```

1 def main():
2     '''

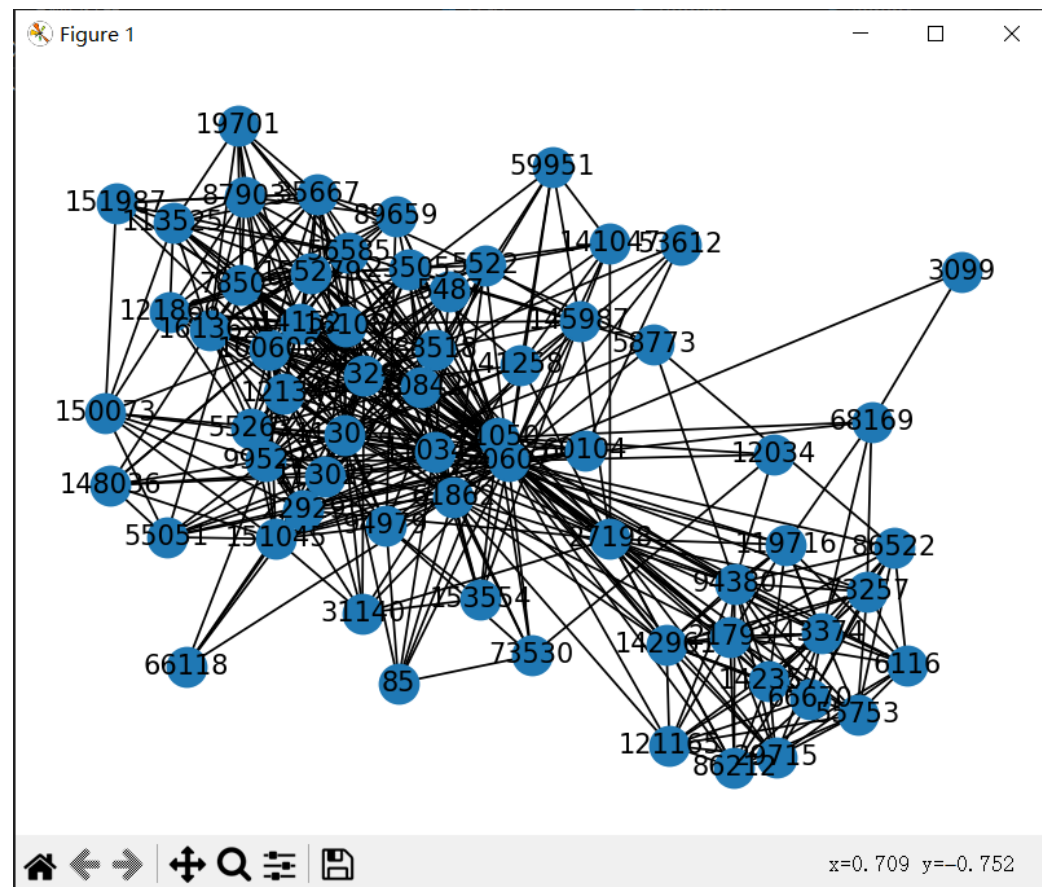
```

```

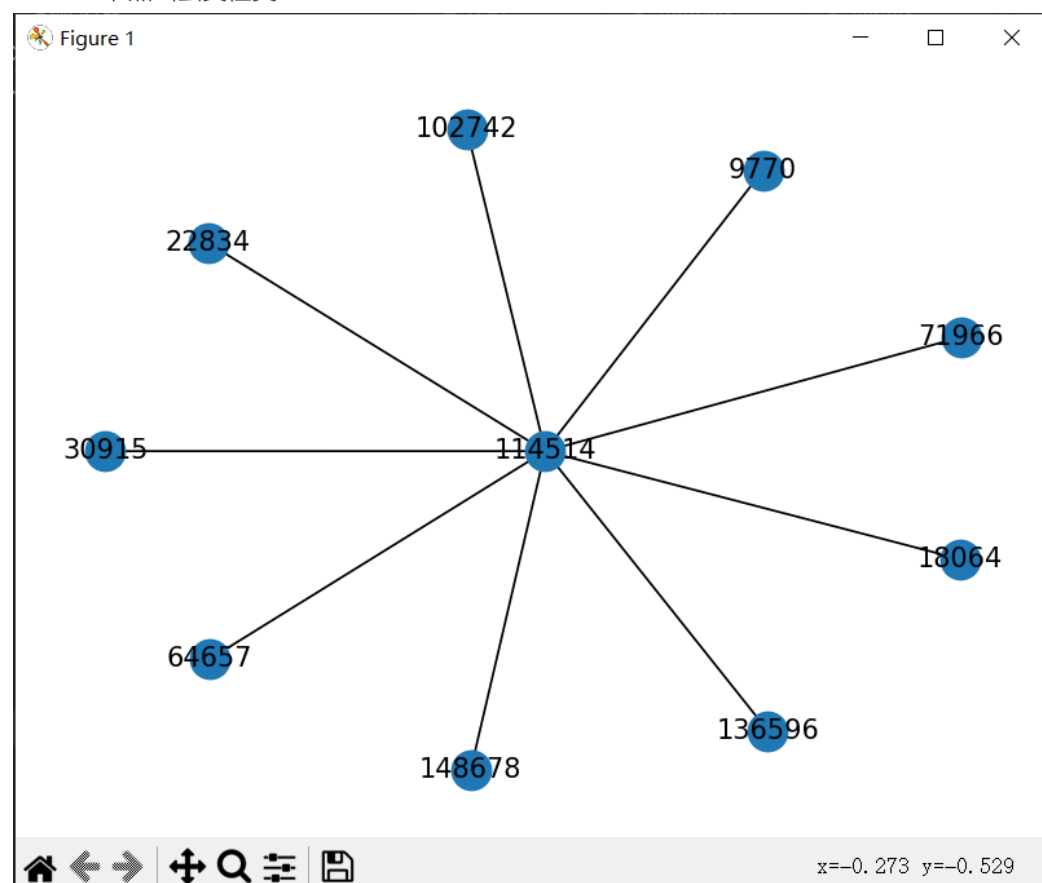
3  测试函数
4  '''
5      graph=load('D:/Project/Python/week4module/GraphStat/NetworkBuilder/sto
      rage_graph.txt')
6      plot_ego(graph,114514)

```

20607节点 (丑)



114514节点 (虽臭但美)

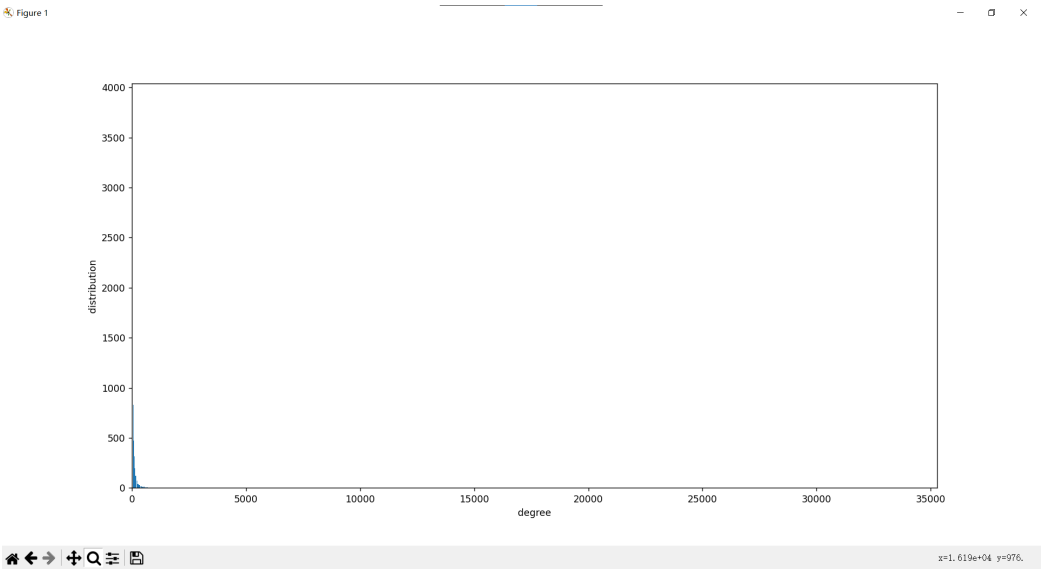


功能：绘制node节点和它的所有邻居

plotdgree\_distribution()

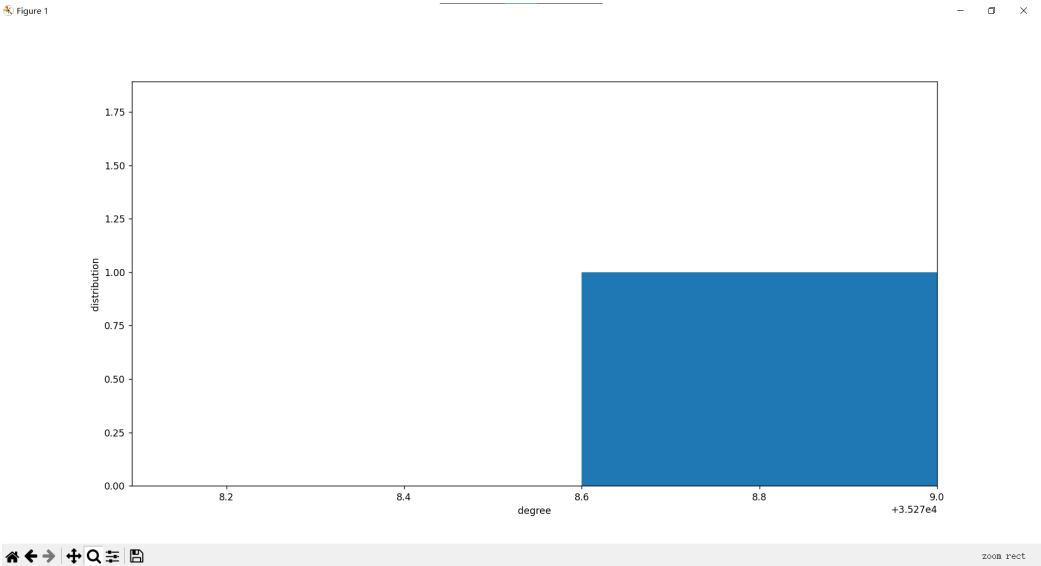
```
1 def main():
2     '''
3     测试函数
4     '''
5     node=load('d:/Project/Python/week4module/GraphStat/NetworkBuilder/storage_node.txt')
6     plotdgree_distribution(node)
7
```

柱形图



放大

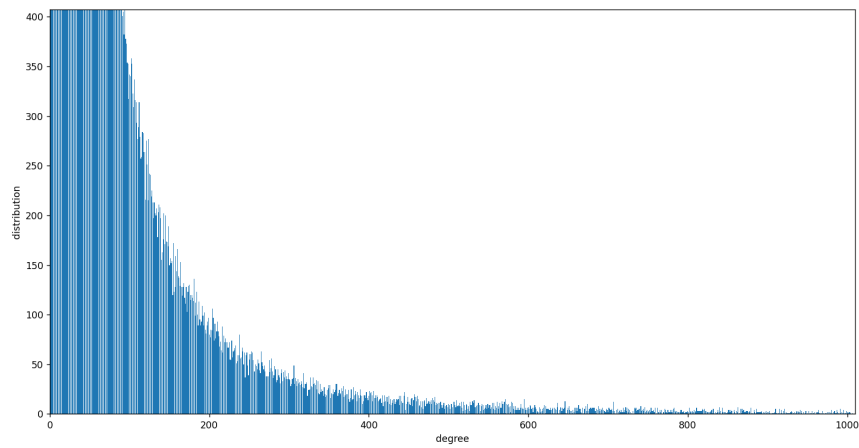
有一个节点有35279个度



前1000包含绝大部分



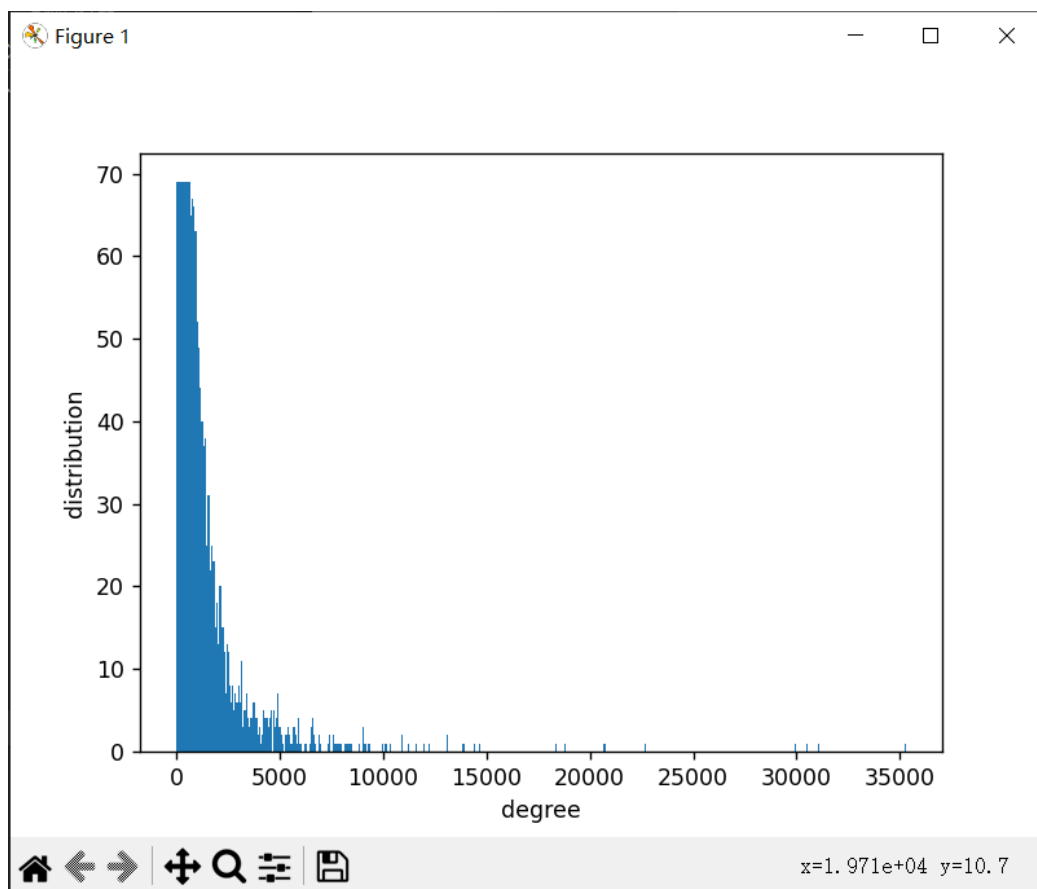
Figure 1



Navigation icons: home, back, forward, search, zoom, and save.

x=327.1 y=83.7

## 直方图



功能：绘制度的分布

横纵轴尺度自适应

## plotnode.py

```
1 '''
2 绘制图中节点属性的统计结果
3 '''
4 from tqdm import tqdm
5 import matplotlib.pyplot as plt
6 import pickle
7
8 def cal_attr_distribution(node_info, attr):
9     '''
```

```

10     计算attr属性的分布
11     返回attr与频数的字典
12     '''
13     v={}
14     for each in node_info:
15         if node_info[each][attr] not in v:
16             v.update({node_info[each][attr]:1})
17         else:
18             v[node_info[each][attr]]+=1
19     return v
20
21 def plot_nodes_attr(node_info,attr) :
22     '''
23     ( 观察属性的分布形态 )
24     '''
25     '''v=cal_attr_distribution(node_info,attr)
26     x=sorted(v.keys())
27     y=[v[key] for key in tqdm(x,desc='loading y...')]
28     plt.bar(x,y)
29     plt.xlabel(attr)
30     plt.ylabel('distribution')
31     plt.xlim(0,max(x))#尺度自适应
32     plt.ylim(0,max(y))
33     plt.show()
34     return 1'''
35     v=cal_attr_distribution(node_info,attr)
36     x=v.keys()
37     plt.hist(x,512)
38     plt.xlabel(attr)
39     plt.ylabel('distribution')
40     plt.show()
41     return 1
42
43 def load(file_path):
44     '''
45     将网络加载至内存
46     '''
47     f=open(file_path,'rb')
48     data=pickle.load(f)
49     f.close()
50     print(file_path,' loaded √')
51     return data
52
53 def main():
54     '''
55     测试函数
56     '''
57     node=load('d:/Project/Python/week4module/GraphStat/NetworkBuilder/storage
58     plot_nodes_attr(node,'views')
59
60 if __name__=='__main__':

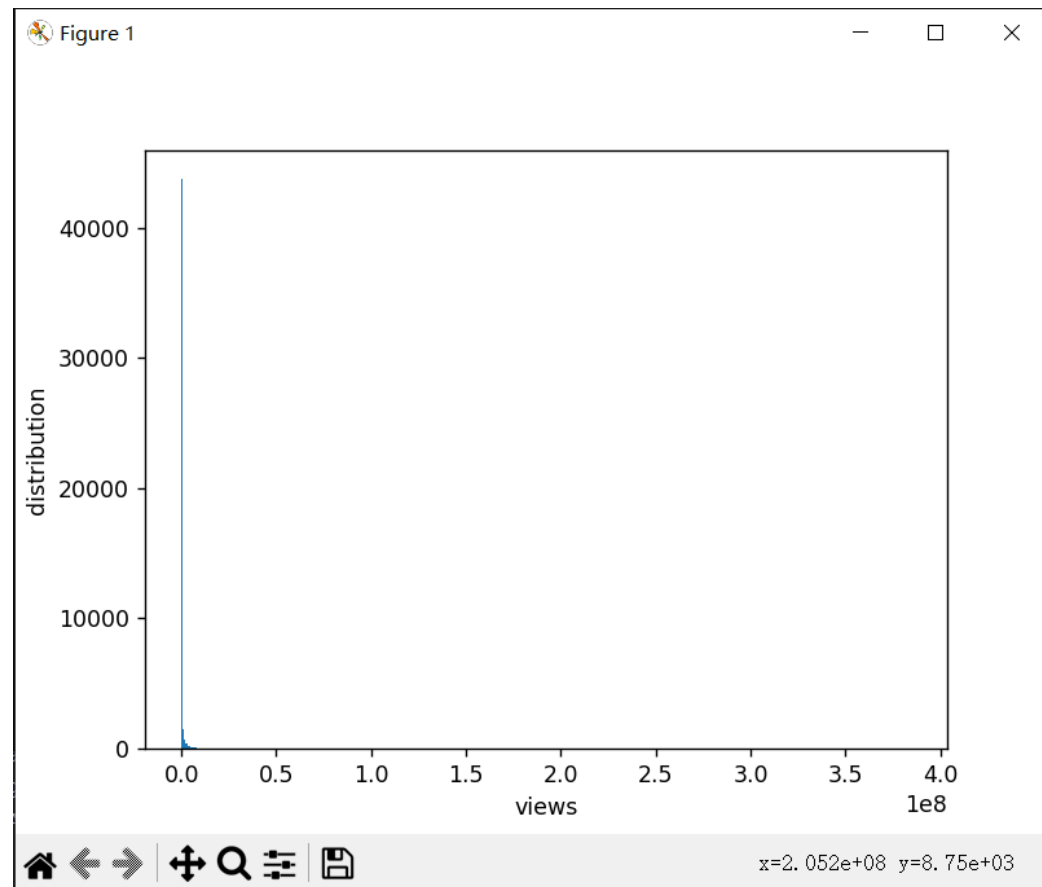
```

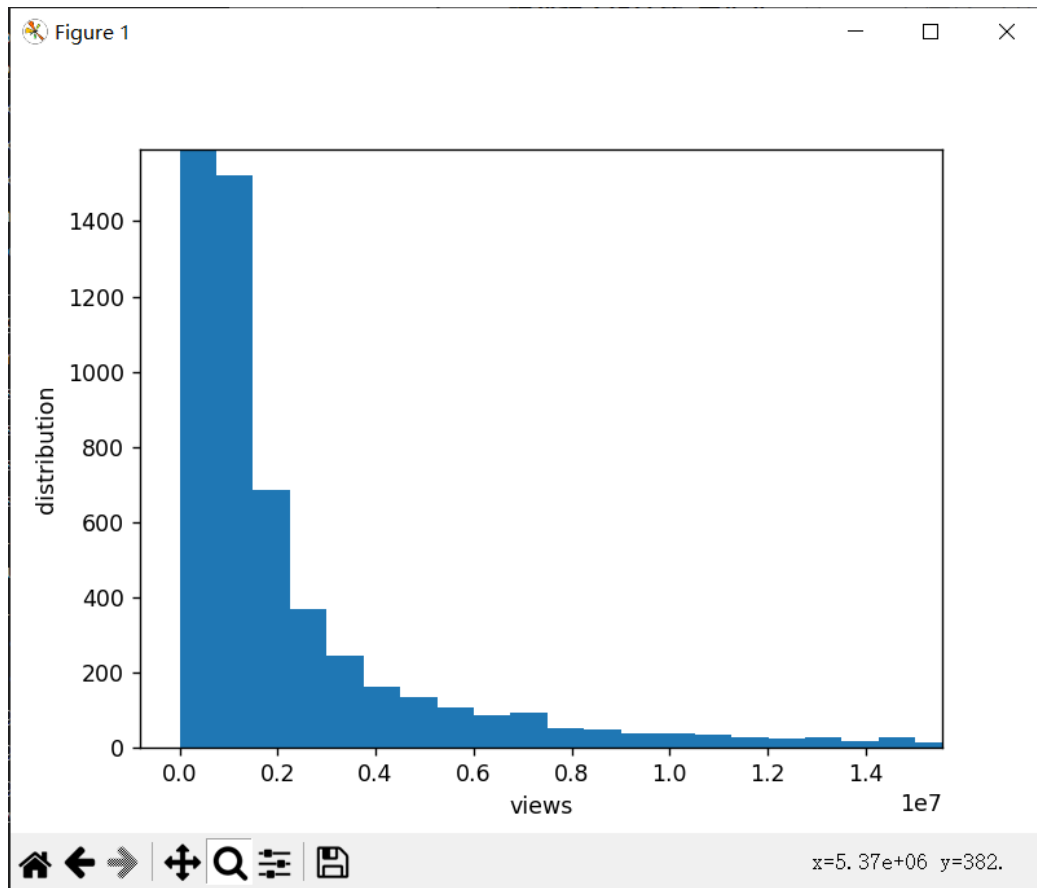
```
61 main()
```

## plot\_nodes\_attr()

```
1 def main():
2     '''
3     测试函数
4     '''
5     node=load('d:/Project/Python/week4module/GraphStat/NetworkBuilder/storage_node.txt')
6     plot_nodes_attr(node,'views')
```

## 直方图





## 附加题

3. 平均度为80.87，绝大多数的节点度在200以内，所以建议抽取度在100以内的节点作为测试对象，使得样本保留总体特征的前提下，便于操作