现代程序设计第4周作业

谢奕飞 20377077

程序包结构

```
GraphStat/
    NetworkBuilder/
        __init__.py
        node.py
            def init_node(info_path,edge_path)
            def get_views(info,node)
            def get_mature(info,node)
            def get_life_time(info,node)
            def get_created_time(info,node)
            def get_updated_time(info,node)
            def get_dead_account(info,node)
            def get_language(info,node)
            def get_link(info,node)
            def print_mode(info,node)
        stat.py
            def get_node_number(node_info)
            def get_edge_number(node_info)
            def cal_average_dgree(node_info)
            def cal_dgree_distribution(node_info)
            def cal_attr_distribution(node_info,attr)
            def load(file_path)
        graph.py
            def init_graph(node_info)
            def save(data,file_name)
            def load(file_path)
            def init_node(info_path,edge_path)
    Visualization/
        plotgraph.py
            def plot_ego(G,node)
            def cal_dgree_distribution(node_info)
            def plotdgree_distribution(node)
            def load(file_path)
        plotnode.py
            def cal_attr_distribution(node_info,attr)
            def plot_nodes_attr(node_info,attr)
            def load(file_path)
```

node.py

```python
import pandas as pd
from tqdm import tqdm

def init_node(info_path,edge_path):
    '''
    从数据文件中加载所有节点及其属性
    返回字典,key 为节点的 ID,值为该节点对应的各属性值(字典)
    '''
    df_info=pd.read_csv(info_path,encoding='utf8')
    node_info=df_info.to_dict(orient='index')
    df_edges=pd.read_csv(edge_path,encoding='utf8')
    for index,edge in tqdm(df_edges.iterrows(),desc='loading edges...'):
        edge1=int(edge['numeric_id_1'])
        edge2=int(edge['numeric_id_2'])
        if 'link' not in node_info[edge1]:
            node_info[edge1].update({'link':{edge2,}})
        else:
            node_info[edge1]['link'].add(edge2)
        if 'link' not in node_info[edge2]:
            node_info[edge2].update({'link':{edge1,}})
        else:
            node_info[edge2]['link'].add(edge1)
    return node_info

def get_views(info,node):
    '''
    获取节点 node 属性
    '''
    return info[node]['views']

def get_mature(info,node):
    '''
    获取节点 node 属性
    '''
    return info[node]['mature']

def get_life_time(info,node):
    '''
    获取节点 node 属性
    '''
    return info[node]['life_time']

def get_created_time(info,node):
    '''
    获取节点 node 属性
    '''
    return info[node]['created_at']

def get_updated_time(info,node):
    '''
    获取节点 node 属性
```

```
    '''
    return info[node]['updated_at']

def get_dead_account(info,node):
    '''
    获取节点 node 属性
    '''
    return info[node]['dead_account']

def get_language(info,node):
    '''
    获取节点 node 属性
    '''
    return info[node]['language']

def get_link(info,node):
    '''
    获取节点 node 属性
    '''
    if 'link' in info[node]:
        return info[node]['link']
    else:
        return {'NONE',}

def print_mode(info,node):
    '''
    显示节点全部信息（利用 format 或者 f 函数）
    '''
    print('-----------------------------------')
    if 'link' in info[node]:
        print('numeric_id: {}\nviews: {}\nmature: {}\nlife_time: {}\ncreated
    else:
        print('numeric_id: {}\nviews: {}\nmature: {}\nlife_time: {}\ncreated
    print('-----------------------------------')

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

init_node()

```
def main():
    '''
    测试函数
    '''
    info_path='D:\Project\Python\week4module\large_twitch_features.csv'
    edge_path='D:\Project\Python\week4module\large_twitch_edges.csv'
    info=init_node(info_path,edge_path)
    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, 1
08187, 123807, 94370, 46243, 136870, 134950, 135216, 151601, 59443, 22333, 82495, 26816, 148674, 73029, 56134, 10441, 16
4047, 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, 1
1, 4619, 17420, 28686, 149006, 157195, 78354, 11288, 90650, 121882, 113692, 76829, 35358, 78367, 56352, 112156, 53788, 2
8703, 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, 625
0, 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, 1220
21, 40101, 28327, 54953, 95914, 137899, 104106, 22189, 124589, 2220, 142512, 130232, 139448, 57532, 42688, 131265, 15840
3, 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, 85
756, 106236, 85758, 2815, 117505, 84226, 144643, 149250, 67333, 115974, 65287, 92424, 113417, 110345, 108809, 111372, 90
374, 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, 1
28328, 27976, 152394, 107854, 154446, 77138, 152402, 61780, 11604, 23378, 146782, 6494, 128864, 43359, 108906, 99179, 13
1436, 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, 2911
8, 56255, 88518, 45001, 165834, 84427, 39372, 140745, 7632, 100305, 6097, 153554, 113620, 53724, 55261, 52703, 79844, 43
498, 3051, 1515, 89069, 153070, 68588, 8176, 165361, 120819, 155127, 73213}}, 2: {'views': 382502, 'mature': 1, 'life_ti
me': 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, 9632
2, 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, 1005
42, 42177, 98503, 101576, 75467, 83660, 73944, 153307, 98524, 29419, 17646, 100593, 86769, 32506, 3323, 78592, 99073, 37
637, 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()

```python
def main():
    '''
    测试函数
    '''
    info_path='D:\Project\Python\week4module\large_twitch_features.csv'
    edge_path='D:\Project\Python\week4module\large_twitch_edges1.csv'
    info=init_node(info_path,edge_path)
    print_mode(info,200)
    print_mode(info,300)
    print_mode(info,400)
    print_mode(info,500)
    print_mode(info,700)
```

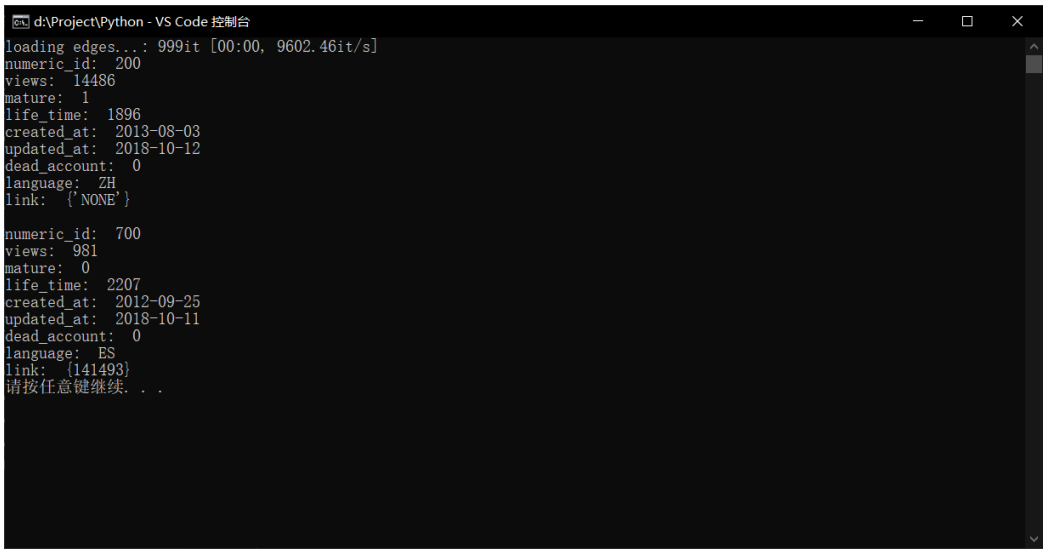
```
d:\Project\Python - VS Code 控制台
loading edges...: 999it [00:00, 9641.30it/s]
-----------------------------------
numeric_id: 200
views: 14486
mature: 1
life_time: 1896
created_at: 2013-08-03
updated_at: 2018-10-12
dead_account: 0
language: ZH
link: NONE
-----------------------------------
-----------------------------------
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()

```python
def main():
    '''
    测试函数
    '''
    info_path='D:\Project\Python\week4module\large_twitch_features.csv'
    edge_path='D:\Project\Python\week4module\large_twitch_edges1.csv'
    ID=200
    info=init_node(info_path,edge_path)
    print('numeric_id: ',ID)
    print('views: ',get_views(info,ID))
    print('mature: ',get_mature(info,ID))
    print('life_time: ',get_life_time(info,ID))
    print('created_at: ',get_created_time(info,ID))
    print('updated_at: ',get_updated_time(info,ID))
    print('dead_account: ',get_dead_account(info,ID))
    print('language: ',get_language(info,ID))
    print('link: ',get_link(info,ID))
    print()
    ID=700
    print('numeric_id: ',ID)
```

```
    print('views: ',get_views(info,ID))
    print('mature: ',get_mature(info,ID))
    print('life_time: ',get_life_time(info,ID))
    print('created_at: ',get_created_time(info,ID))
    print('updated_at: ',get_updated_time(info,ID))
    print('dead_account: ',get_dead_account(info,ID))
    print('language: ',get_language(info,ID))
    print('link: ',get_link(info,ID))
```



（以large_twitch_edges.csv前1000行数据为例）

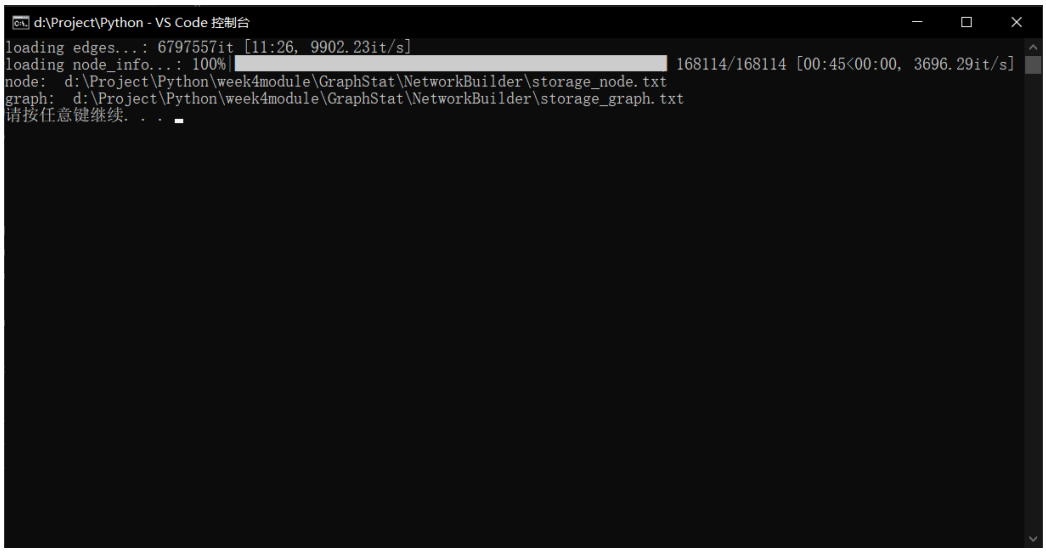
功能：获得某个节点的XXX属性

graph.py

```
import pandas as pd
from tqdm import tqdm
import pickle
import os
import sys
import networkx as nx

def init_graph(node_info):
    '''
    构建网络
    '''
    G=nx.Graph()
    for each in tqdm(node_info,desc='loading node_info...'):
        if 'link' not in node_info[each]:
            continue
        for link in node_info[each]['link']:
            G.add_edge(each,link)
    return G

def save(data,file_name):
    '''
    序列化信息
    '''
    file_path=str(sys.argv[0])[:-len(os.path.basename(sys.argv[0]))]+file_na
```

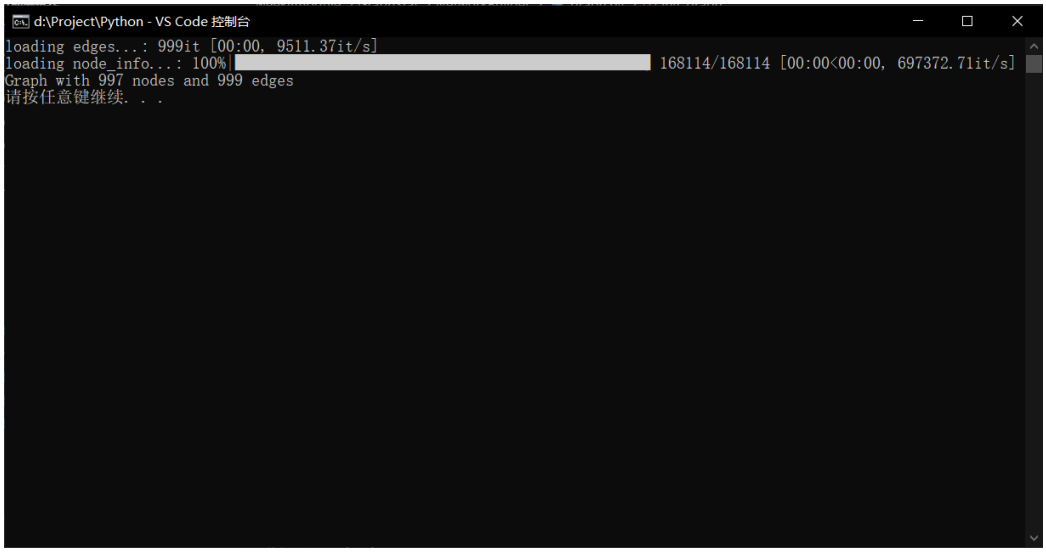
```
    f=open(file_path,'wb')
    pickle.dump(data,f,0)
    f.close
    return file_path

def load(file_path):
    '''
    将序列化信息加载至内存
    '''
    f=open(file_path,'rb')
    data=pickle.load(f)
    f.close()
    print(file_path,' loaded √')
    return data

def init_node(info_path,edge_path):
    '''
    从数据文件中加载所有节点及其属性
    返回字典,key 为节点的 ID,值为该节点对应的各属性值(字典)
    '''
    df_info=pd.read_csv(info_path,encoding='utf8')
    node_info=df_info.to_dict(orient='index')
    df_edges=pd.read_csv(edge_path,encoding='utf8')
    for index,edge in tqdm(df_edges.iterrows(),desc='loading edges...'):
        edge1=int(edge['numeric_id_1'])
        edge2=int(edge['numeric_id_2'])
        if 'link' not in node_info[edge1]:
            node_info[edge1].update({'link':{edge2,}})
        else:
            node_info[edge1]['link'].add(edge2)
        if 'link' not in node_info[edge2]:
            node_info[edge2].update({'link':{edge1,}})
        else:
            node_info[edge2]['link'].add(edge1)
    return node_info

def main():
    '''
    测试函数
    '''
    node=load('d:/Project/Python/week4module/GraphStat/NetworkBuilder/storag
    graph=load('D:/Project/Python/week4module/GraphStat/NetworkBuilder/stora
    print()

```

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



init_graph()

```
def main():
    '''
    测试函数
    '''
    info_path='D:\Project\Python\week4module\large_twitch_features.csv'
    edge_path='D:\Project\Python\week4module\large_twitch_edges1.csv'
    node_info=init_node(info_path,edge_path)
    graph=init_graph(node_info)
    print(graph)
```

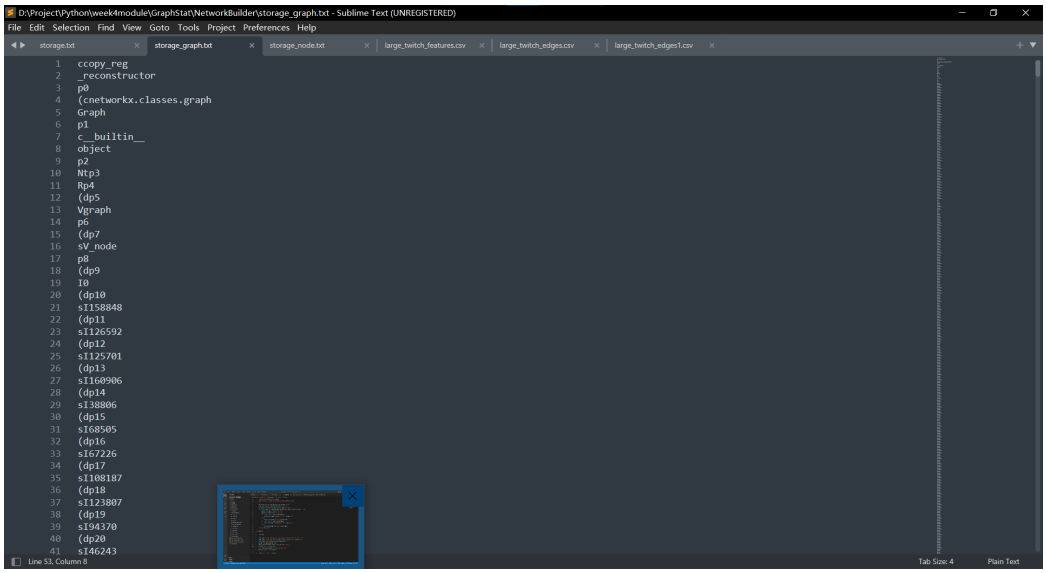


（以large_twitch_edges.csv前1000行数据为例）

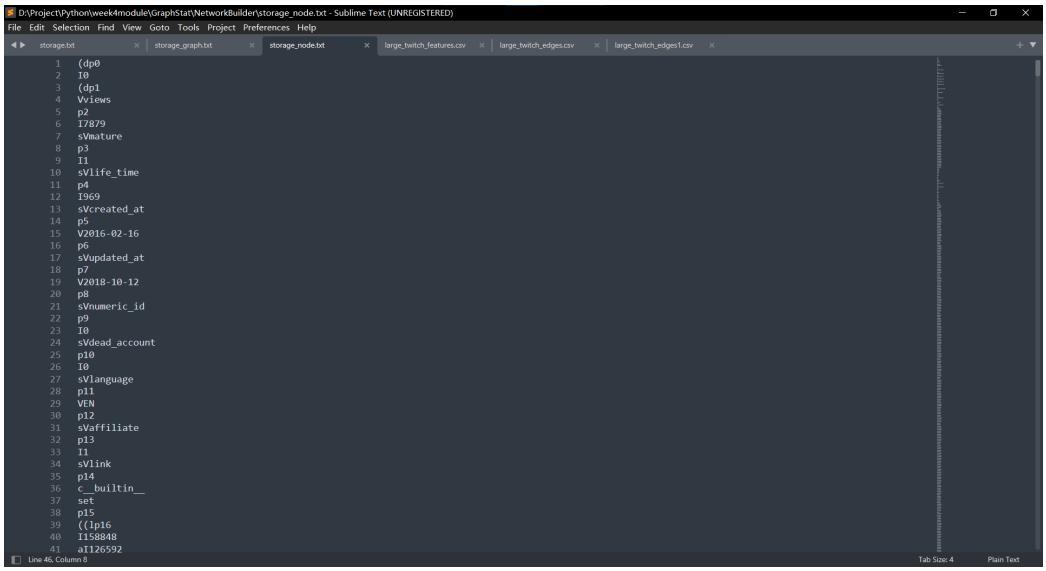
功能：从node_info的link中读取边的信息，构建节点与所连节点的字典

save()

storage_graph.txt



storage_node.txt



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()

```
def main():
    '''
    测试函数
    '''
    node=load('d:/Project/Python/week4module/GraphStat/NetworkBuilder/storag
    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, 1
08187, 123807, 94370, 46243, 136870, 134950, 135216, 151601, 59443, 22333, 82495, 26816, 148674, 73029, 56134, 10441, 16
4047, 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, 1
1, 4619, 17420, 28686, 149006, 157195, 78354, 11288, 121882, 90650, 112156, 113692, 76829, 35358, 56352, 78367, 53788, 2
8703, 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, 625
0, 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, 1
63040, 87271, 152296, 119017, 15080, 83691, 16108, 146669, 8430, 9455, 112881, 128241, 118516, 86266, 119034, 85756, 106
236, 85758, 2815, 117505, 84226, 144643, 149250, 67333, 115974, 142087, 65287, 92424, 113417, 110345, 108809, 111372, 90
374, 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, 1
28328, 27976, 152394, 107854, 154446, 77138, 152402, 61780, 11604, 23378, 146782, 6494, 128864, 43359, 108906, 99179, 13
1436, 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, 2911
8, 56255, 88518, 45001, 165834, 84427, 39372, 140745, 7632, 100305, 6097, 153554, 113620, 53724, 55261, 52703, 79844, 43
498, 3051, 1515, 89069, 153070, 68588, 8176, 165361, 120819, 155127, 73213}}, 2: {'views': 382502, 'mature': 1, 'life_ti
me': 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
2, 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, 1005
42, 42177, 98503, 101576, 75467, 83660, 73944, 153307, 98524, 29419, 17646, 100593, 86769, 32506, 3323, 78592, 99073, 37
637, 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
```

```python
def main():
    '''
    测试函数
    '''
    graph=load('D:/Project/Python/week4module/GraphStat/NetworkBuilder/storage_graph.txt')
    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

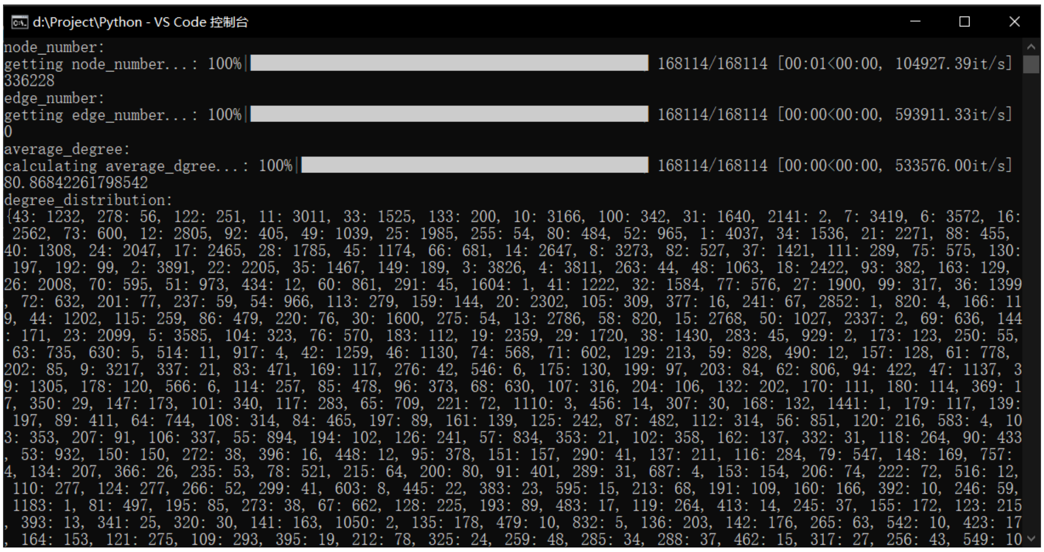
stat.py

```python
from tqdm import tqdm
import pickle

def get_node_number(node_info):
    '''
    计算节点数
    '''
    node=set()
    for each in tqdm(node_info,desc='getting node_number...'):
        node.add(each)
        if 'link' in node_info[each]:
            node.add(link for link in node_info[each]['link'])
```

```
    return len(node)

def get_edge_number(node_info):
    '''
    计算边数
    '''
    edge=set()
    for each in tqdm(node_info,desc='getting edge_number...'):
        if 'link' in node_info[each]['link']:
            for link in node_info[each]['link']:
                edge.add(set(each,link))
        else:
            continue
    return len(edge)

def cal_average_dgree(node_info):
    '''
    计算网络中的平均度
    '''
    dgrees=0
    node_number=0
    for each in tqdm(node_info,desc='calculating average_dgree...'):
        if 'link' in node_info[each]:
            dgrees+=len(node_info[each]['link'])
        node_number+=1
    return dgrees/node_number

def cal_dgree_distribution(node_info):
    '''
    计算网络的度分布
    返回度与频数的字典
    '''
    dgrees={}
    for each in node_info:
        if 'link' in node_info[each]:
            dgree=len(node_info[each]['link'])
            if dgree not in dgrees:
                dgrees.update({dgree:1})
            else:
                dgrees[dgree]+=1
    return dgrees

def cal_attr_distribution(node_info,attr):
    '''
    计算attr属性的分布
    返回attr与频数的字典
    '''
    v={}
    for each in node_info:
        if node_info[each][attr] not in v:
            v.update({node_info[each][attr]:1})
```

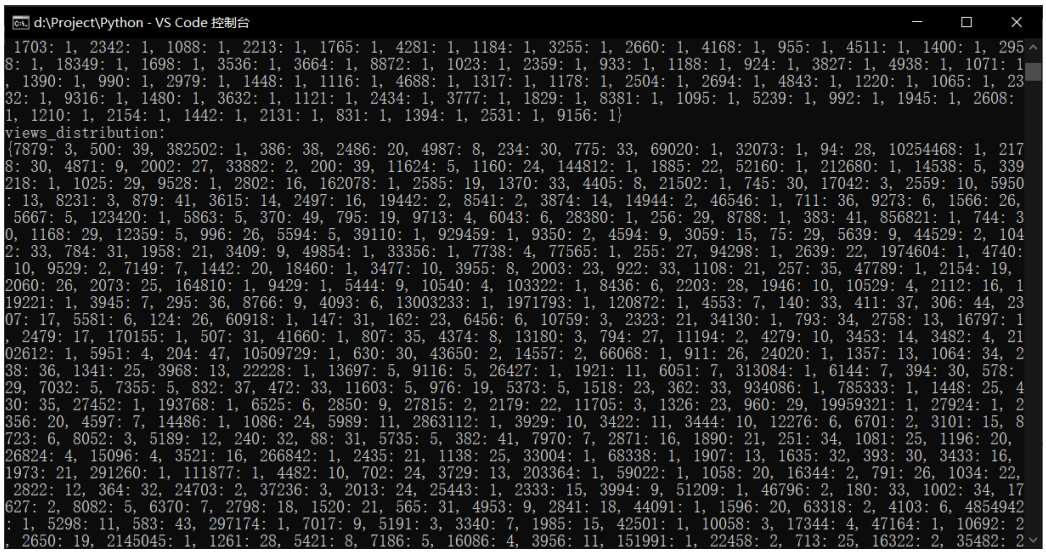
```
        else:
            v[node_info[each][attr]]+=1
    return v

def load(file_path):
    '''
    将网络加载至内存
    '''
    f=open(file_path,'rb')
    data=pickle.load(f)
    f.close()
    print(file_path,' loaded √')
    return data

def main():
    '''
    测试函数
    '''
    node=load('d:/Project/Python/week4module/GraphStat/NetworkBuilder/storag
    print('node_number: ')
    print(get_node_number(node))
    print('edge_number: ')
    print(get_edge_number(node))
    print('average_degree: ')
    print(cal_average_dgree(node))
    print('degree_distribution: ')
    print(cal_dgree_distribution(node))
    print('views_distribution: ')
    print(cal_attr_distribution(node,'views'))


if __name__=='__main__':
```

main()   main()



```
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
```

…



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)	返回平均度
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

```
import networkx as nx
import matplotlib.pyplot as plt
import pickle
from tqdm import tqdm

def plot_ego(G,node):
    '''
    （附加：使用 networkx 库中的布局算法可视化结构，
    注意避免结构太大，复杂可能导致绘制失败，或者杂乱。）
    绘制节点的局部网络（找一些度大小合适的节点尝试。）
    '''
    G_neighbor=nx.ego_graph(G,node)
    nx.draw(G_neighbor,with_labels=True)
    plt.show()
    return 1

def cal_dgree_distribution(node_info):
    '''
    计算网络的度分布
    返回度与频数的字典
    '''
    dgrees={}
    for each in node_info:
        if 'link' in node_info[each]:
            dgree=len(node_info[each]['link'])
```

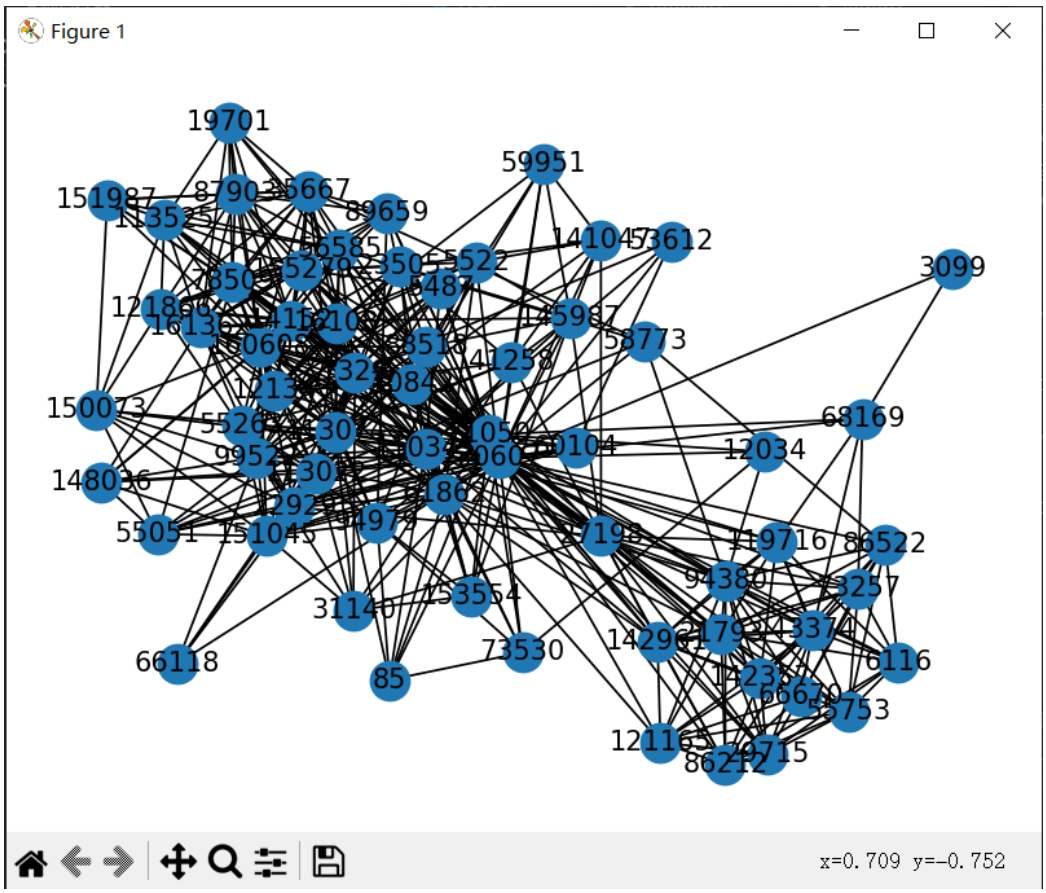
```
            if dgree not in dgrees:
                dgrees.update({dgree:1})
            else:
                dgrees[dgree]+=1
    return dgrees

def plotdgree_distribution(node):
    '''
    （观察度分布的形态）
    度的分布图
    '''
    dgree=cal_dgree_distribution(node)
    '''x=sorted(dgree.keys())
    y=[dgree[key] for key in tqdm(x,desc='loading y...')]
    plt.bar(x,y)
    plt.xlabel('degree')
    plt.ylabel('distribution')
    plt.xlim(0,max(x))#尺度自适应
    plt.ylim(0,max(y))
    plt.show()
    return 1'''
    x=dgree.keys()
    plt.hist(x,512)
    plt.xlabel('degree')
    plt.ylabel('distribution')
    plt.show()
    return 1

def load(file_path):
    '''
    将网络加载至内存
    '''
    f=open(file_path,'rb')
    data=pickle.load(f)
    f.close()
    print(file_path,' loaded √')
    return data

def main():
    '''
    测试函数
    '''
    node=load('d:/Project/Python/week4module/GraphStat/NetworkBuilder/storag
    plotdgree_distribution(node)

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

plot_ego()

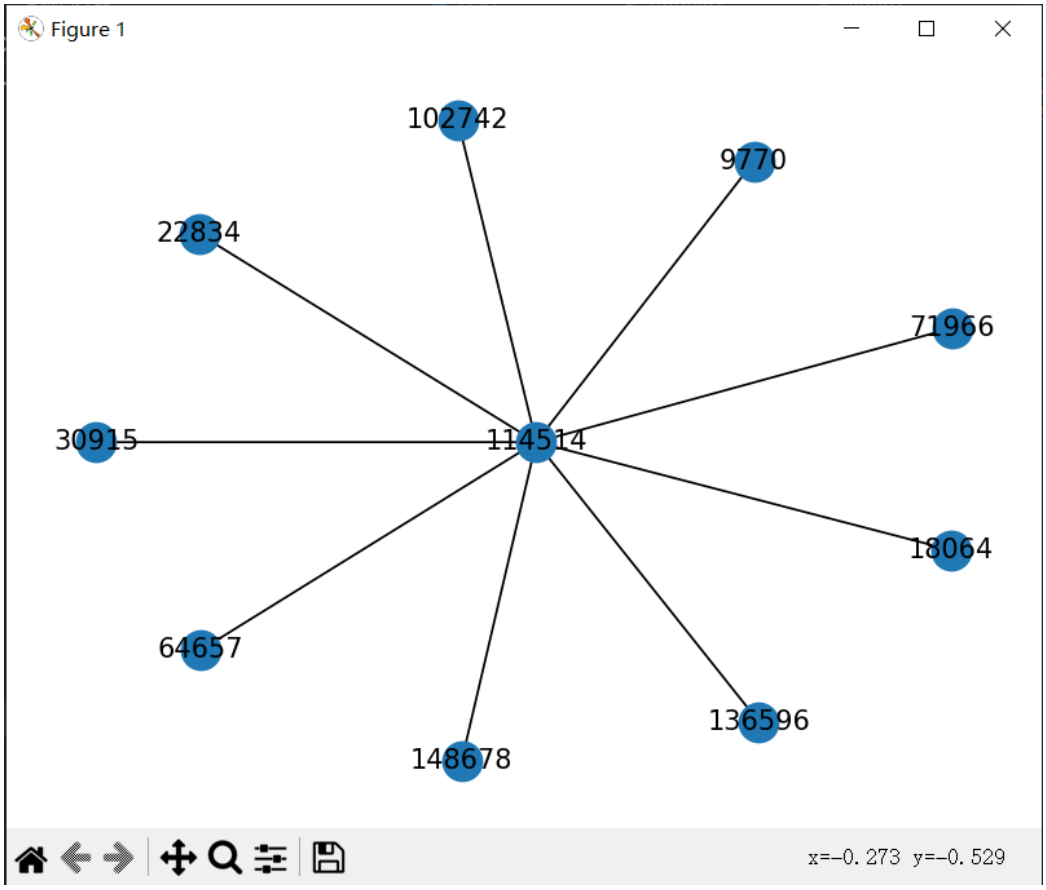
```
def main():
    '''
```

```
    测试函数
    '''
    graph=load('D:/Project/Python/week4module/GraphStat/NetworkBuilder/storage_graph.txt')
    plot_ego(graph,114514)
```

20607节点（丑



114514节点（虽臭但美

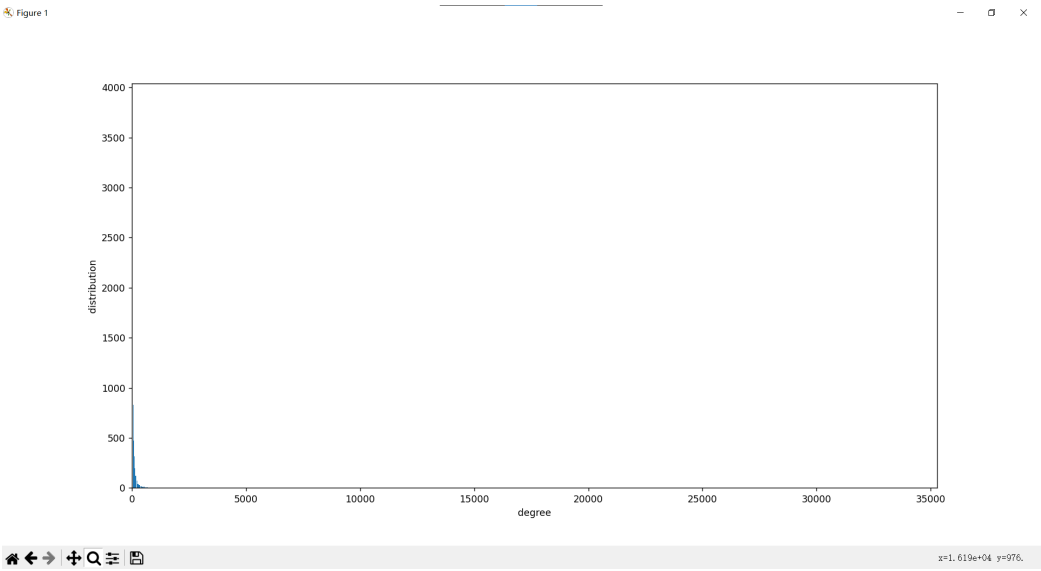


功能：绘制node节点和它的所有邻居

plotdgree_distribution()

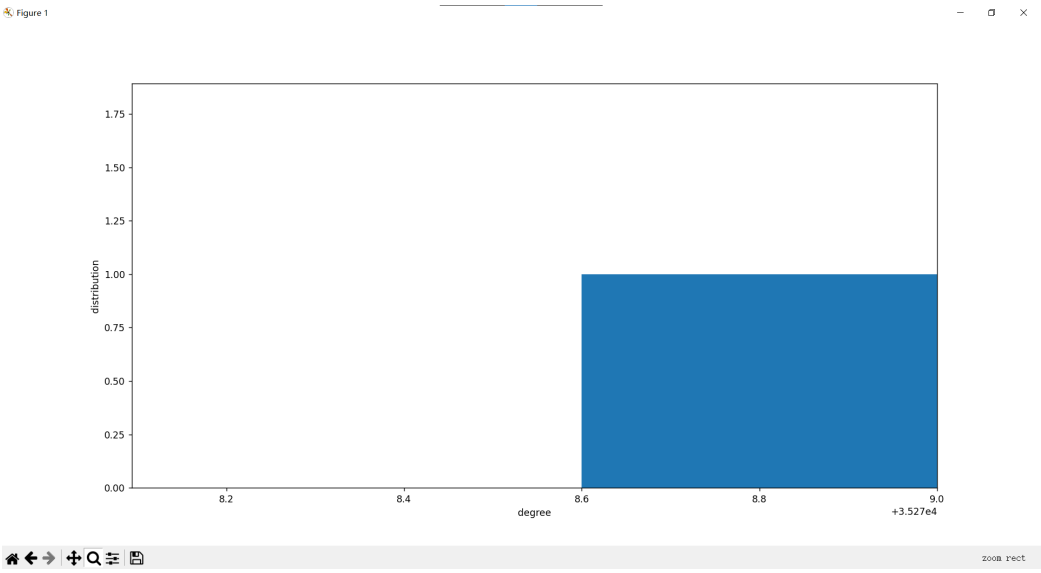
```
def main():
    '''
    测试函数
    '''
    node=load('d:/Project/Python/week4module/GraphStat/NetworkBuilder/storage_node.txt')
    plotdgree_distribution(node)

```

柱形图

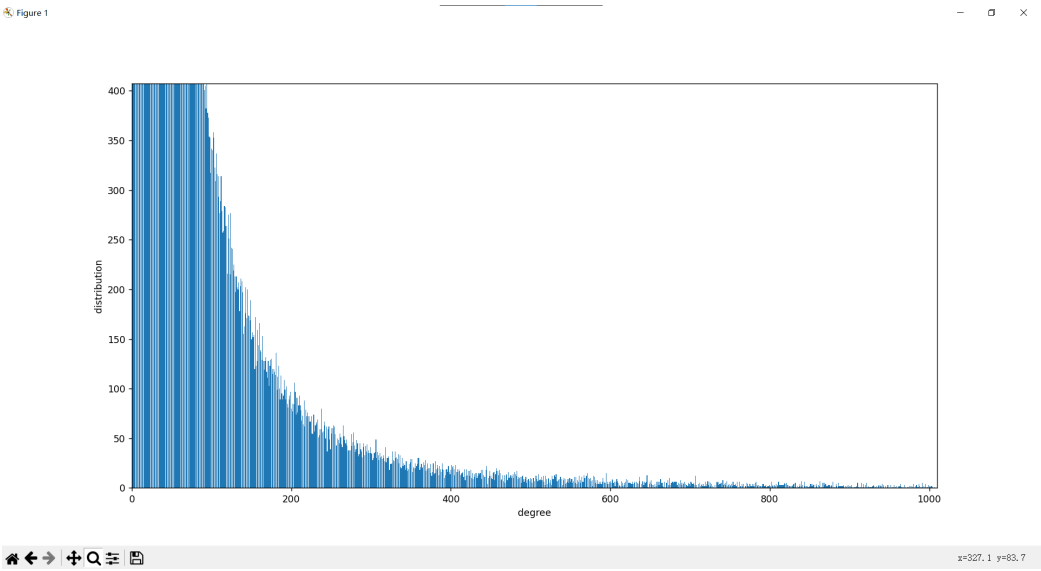


放大

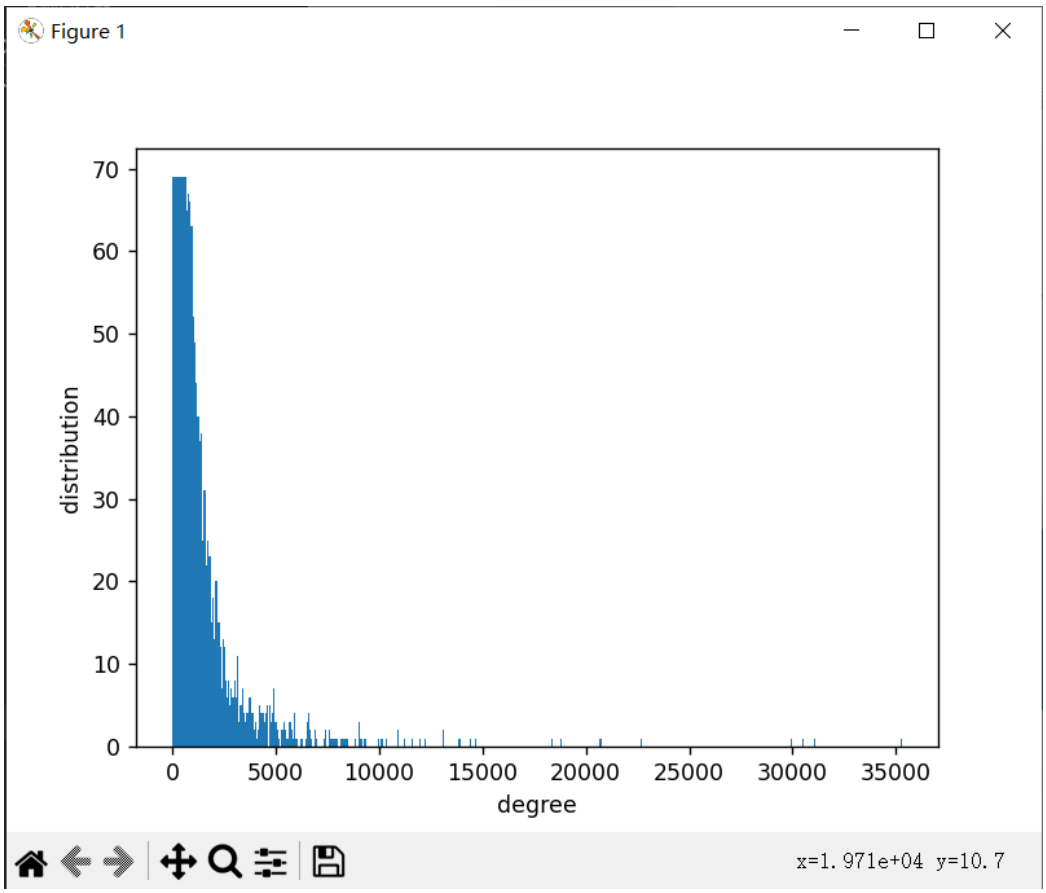
有一个节点有35279个度



前1000包含绝大部分



直方图



功能：绘制度的分布

横纵轴尺度自适应

plotnode.py

```
'''
绘制图中节点属性的统计结果
'''
from tqdm import tqdm
import matplotlib.pyplot as plt
import pickle

def cal_attr_distribution(node_info,attr):
    '''
```

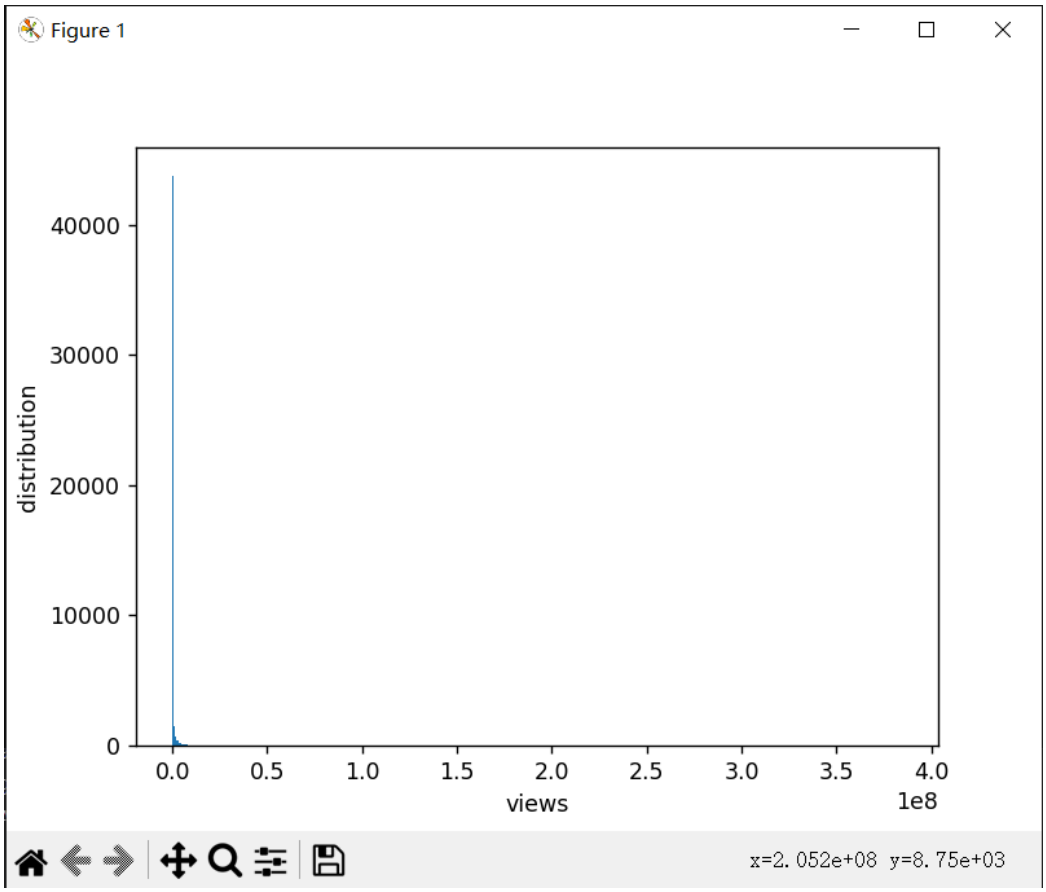
```
    计算attr属性的分布
    返回attr与频数的字典
    '''
    v={}
    for each in node_info:
        if node_info[each][attr] not in v:
            v.update({node_info[each][attr]:1})
        else:
            v[node_info[each][attr]]+=1
    return v

def plot_nodes_attr(node_info,attr) :
    '''
    （观察属性的分布形态）
    '''
    '''v=cal_attr_distribution(node_info,attr)
    x=sorted(v.keys())
    y=[v[key] for key in tqdm(x,desc='loading y...')]
    plt.bar(x,y)
    plt.xlabel(attr)
    plt.ylabel('distribution')
    plt.xlim(0,max(x))#尺度自适应
    plt.ylim(0,max(y))
    plt.show()
    return 1'''
    v=cal_attr_distribution(node_info,attr)
    x=v.keys()
    plt.hist(x,512)
    plt.xlabel(attr)
    plt.ylabel('distribution')
    plt.show()
    return 1

def load(file_path):
    '''
    将网络加载至内存
    '''
    f=open(file_path,'rb')
    data=pickle.load(f)
    f.close()
    print(file_path,' loaded √')
    return data

def main():
    '''
    测试函数
    '''
    node=load('d:/Project/Python/week4module/GraphStat/NetworkBuilder/storag
    plot_nodes_attr(node,'views')

if __name__=='__main__':
```

```
    main()
```

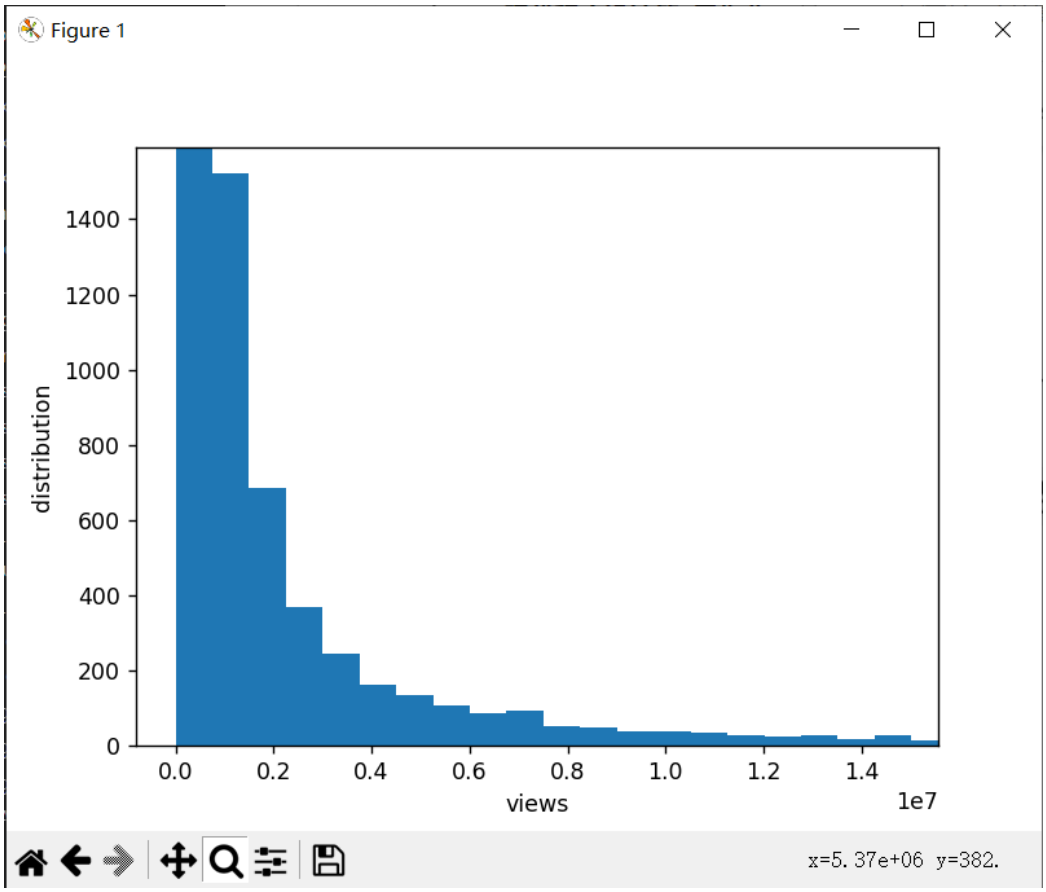
plot_nodes_attr()

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

直方图



放大



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