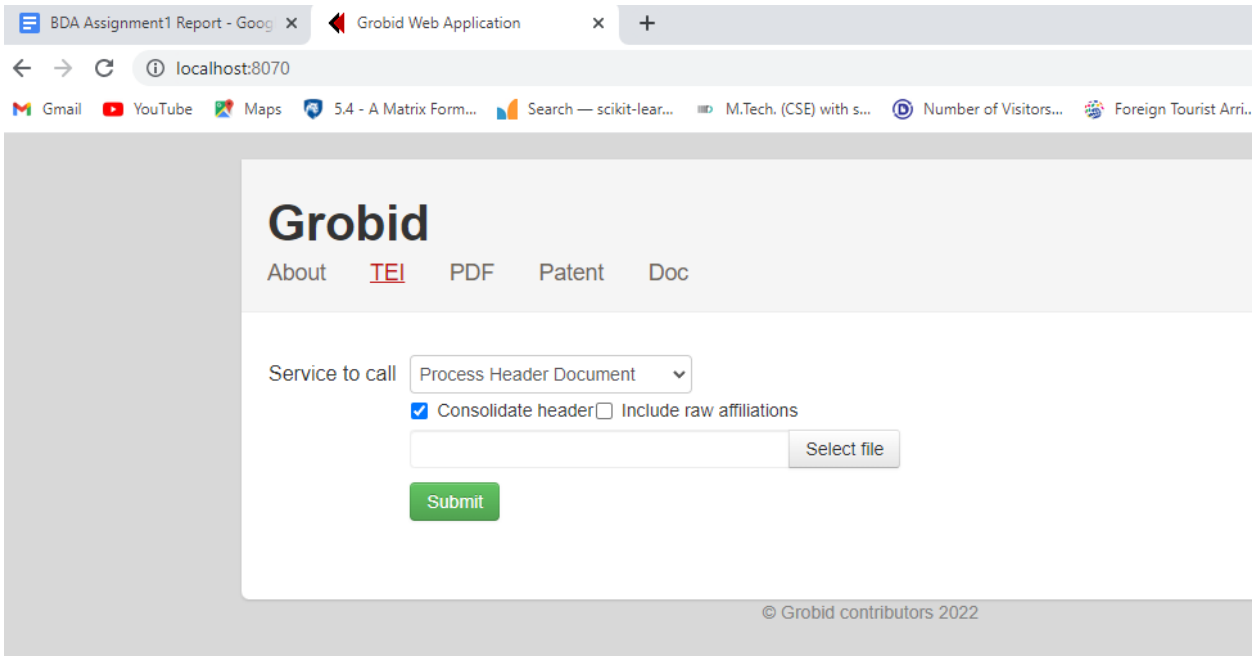
Approach:

Extract Titles from D1:
Merge the titles and subtitles into a single column

Extract Titles from D2:

1. We have used **Grobid** for this purpose
    a. Install Grobid by pulling its docker image
        ====> docker pull lfoppiano/grobid:0.7.2
    b. Run docker image
        ====> docker run -t --rm -p 8070:8070 lfoppiano/grobid:0.7.2

    c. Install pygrobid library in python so that we can access grobid
    d. Access grobid



Grobid is now active on port 8070

e. Fetch titles

===>import library

```
[ ]: from grobid_client.grobid_client import GrobidClient
```

```
[1]: from grobid_client_python.grobid_client.grobid_client import GrobidClient
```

===> make sure we can reach Grobid

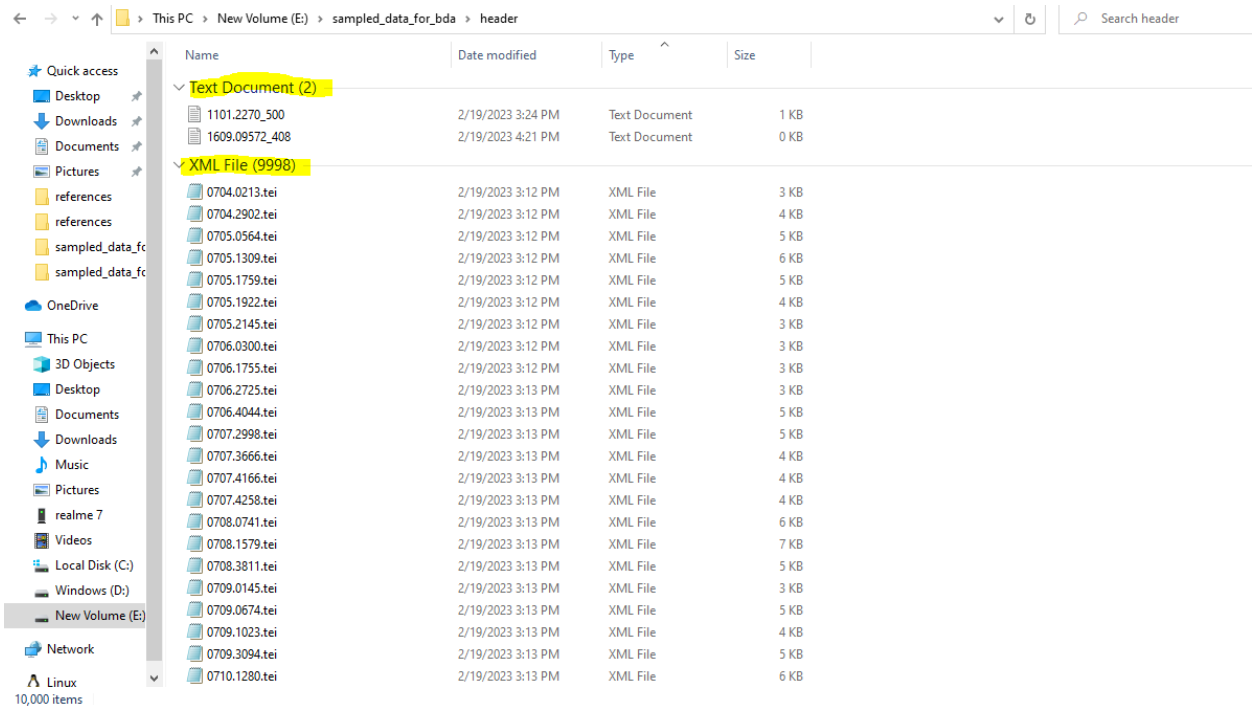
```
dire = 'E:/sampled_data_for_bda/sampled_data_for_bda/2102.07623.pdf'

client = GrobidClient(grobid_server='localhost',
                      sleep_time=5,
                      timeout=70,
                      config_path="C:/Users/hp/AppData/Local/Programs/Python/Python311/Lib/site-packages/grobid_client_pyth
                      check_server=True)
```

```
GROBID server is up and running
```

===>extract files

```
import os
new_path = 'E:/TESTING BDA/sampled_data_for_bda/'
client.process("processHeaderDocument", new_path,
               n=10*3,
               consolidate_header=True,
               include_raw_citations=False,
               include_raw_affiliations=True,
               verbose=True,
               output = os.path.join(new_path, 'header2'))
```

f. We will get XML files in return



Total 9998 xmls were retrieved means we got 9998 probable titles and 2 text files indicate no titles. There are also some black titles retrieved from XML which will be dealt with in further steps. Lets see how the XML looks like.

```
"1.0" encoding="UTF-8"?>
="preserve" xmlns="http://www.tei-c.org/ns/1.0"
p://www.w3.org/2001/XMLSchema-instance"
tion="http://www.tei-c.org/ns/1.0 https://raw.githubusercontent.com/kermitt2/grobid/master/grobid-home/schemas/xsd/Grobid.xsd"
http://www.w3.org/1999/xlink">
ader xml:lang="en">
  <fileDesc>
         <titleStmt>
                <title level="a" type="main">Recommending Related Papers Based on Digital Library Access Records</title>
         </titleStmt>
         <publicationStmt>
                <publisher/>
                <availability status="unknown"><licence/></availability>
         </publicationStmt>
         <sourceDesc>
                <biblStruct>
                       <analytic>
                              <author>
                                     <persName><forename type="first">Stefan</forename><surname>Pohl</surname></persName>
                                     <affiliation key="aff0">
                                            <note type="raw_affiliation">Cornell University Ithaca, NY, USA</note>
                                            <orgName type="institution">Cornell University Ithaca</orgName>
                                            <address>
                                                   <region>NY</region>
                                                   <country key="US">USA</country>
                                            </address>
                                     </affiliation>
                              </author>
                              <author>
                                     <persName><forename type="first">Filip</forename><surname>Radlinski</surname></persName>
                                     <affiliation key="aff1">
                                            <note type="raw_affiliation">Cornell University Ithaca, NY, USA</note>
                                            <orgName type="institution">Cornell University Ithaca</orgName>
                                            <address>
                                                   <region>NY</region>
                                                   <country key="US">USA</country>
```

The highlighted part is the title we need.

g. Use the Beautifulsoop library to extract titles from the XML file

```
import os
from bs4 import BeautifulSoup
import nltk
dire = 'C:/Users/91775/Downloads/header/'
i=1;
count = 0
title_ = {}
for filename in os.listdir(dire):
    print(filename)
    f = os.path.join(dire, filename) # joining the file name with file..
    file1 = open(f,errors = 'ignore')
    soup = file1.read()# reading the file 1..
    bs = BeautifulSoup(soup,"html.parser") # parsing the text in file1..
    #text = bs.find('text').text # finding the text with in the given file
    title = bs.find('title').text # finding the text with <Title> in the given file
   # texti = title+text # cocatenating the <Title> and  string..
    file1.close();
    print(count)

    print("-------------------------"+"file after extracting text from tags <Title> and <
    print(title)
    if(title == ""):
        count = count+1;
    title_[title] = filename

    count +=1

print(count)
```

```
0704.0213.tei.xml
0
-------------------------file after extracting text from tags <Title> and <Text>:0704.0213.tei.xml----------------------
Geometric Complexity Theory V: On deciding nonvanishing of a generalized Littlewood-Richardson coefficient Dedicated to Sri
amakrishna
0704.2902.tei.xml
1
-------------------------file after extracting text from tags <Title> and <Text>:0704.2902.tei.xml----------------------
Recommending Related Papers Based on Digital Library Access Records
0705.0564.tei.xml
2
-------------------------file after extracting text from tags <Title> and <Text>:0705.0564.tei.xml----------------------
Rate Bounds for MIMO Relay Channels Using Message Splitting
0705.1309.tei.xml
3
-------------------------file after extracting text from tags <Title> and <Text>:0705.1309.tei.xml----------------------

0705.1759.tei.xml
```

h. Store the titles in the form of a dictionary with the key being the extracted title and the value being the filename.

2. For the remaining titles we have used **pdfTitle** library from python
   This has reduced the error rate in fetching titles by another 1%

```
import os
import pdftitle

count=0
dire = 'E:/sampled_data_for_bda/sampled_data_for_bda/'
for filename in os.listdir(dire):
    try:
        fname = os.path.join(dire, filename)
        title = pdftitle.get_title_from_file(fname)
        print(title + " ===> " + filename)
        count=count+1
    except:
        continue

print(count)
```

Outcome:

```
FreeBSD Mandatory Access Control Usage for Implementing Enterprise Security Policies ===> 0706.1755.pdf
The Complexity of Determining Existence a Hamiltonian Cycle is ===> 0706.2725.pdf
PSPACE Bounds for Rank-1 Modal Logics ===> 0706.4044.pdf
BuildingaCooperativeCommunicationsSystem ===> 0707.2998.pdf
KINEMATIC ANALYSIS OF A NEW PARALLEL MACHINE TOOL: THE ORTHOGLIDE ===> 0707.3666.pdf
Unfolding Convex Polyhedra via Quasigeodesics ===> 0707.4258.pdf
Characterising Web Site Link Structure ===> 0708.0741.pdf
Homogeneous temporal activity patterns in a large online communication space ===> 0708.1579.pdf
AN EXHAUSTIVE STUDY OF THE WORKSPACE TOPOLOGIES OF ALL 3R ORTHOGONAL MANIPULATORS WITH GEOMETRIC SIMPLIFICATIONS ===>
0708.3811.pdf
Estimating Random Variables from Random Sparse Observations ===> 0709.0145.pdf
```

In the end we have 9896 titles which is **98.96% accuracy**

Extract References from D2:

Again we have Grobid for extracting references from D2 with the following steps

a. Run docker image
b. Access grobid using pygrobid
c. Fetch references

```
import os
new_path = 'E:/TESTING BDA/sampled_data_for_bda/'
client.process("processReferences", new_path,
        n=10*3,
        consolidate_header=True,
        verbose=True,
        output = os.path.join(new_path, 'references2'))
```

```
1512.08279.pdf
1 files to process in current batch
1911.06565.pdf
1 files to process in current batch
Processing of E:/TESTING BDA/sampled_data_for_bda/\1911.06565.pdf failed with error 408 , None
1911.06714.pdf
1 files to process in current batch
Processing of E:/TESTING BDA/sampled_data_for_bda/\1911.06714.pdf failed with error 408 , None
2003.09451.pdf
1 files to process in current batch
Processing of E:/TESTING BDA/sampled_data_for_bda/\2003.09451.pdf failed with error 408 , None
```

d. We will get XML files in return

```
<biblStruct xml:id="b2">
        <analytic>
                <title level="a" type="main">Accurately interpreting clickthrough data as implicit feedback</title>
                <author>
                        <persName><forename type="first">T</forename><surname>Joachims</surname></persName>
                </author>
                <author>
                        <persName><forename type="first">L</forename><surname>Granka</surname></persName>
                </author>
                <author>
                        <persName><forename type="first">B</forename><surname>Pang</surname></persName>
                </author>
                <author>
                        <persName><forename type="first">H</forename><surname>Hembrooke</surname></persName>
                </author>
                <author>
                        <persName><forename type="first">G</forename><surname>Gay</surname></persName>
                </author>
        </analytic>
        <monogr>
                <title level="m">SIGIR</title>
                        <imprint>
                <date type="published" when="2005">2005</date>
                </imprint>
        </monogr>
</biblStruct>

<biblStruct xml:id="b3">
        <analytic>
                <title level="a" type="main">On the recommending of citations for research papers</title>
                <author>
                        <persName><forename type="first">S</forename><forename type="middle">M</forename><surname>Mcnee</surname></
                </author>
                <author>
                        <persName><forename type="first">I</forename><surname>Albert</surname></persName>
```

The highlighted parts represent the references which we need to extract.
We extracted 9939 references which is **99.39% accuracy**

e. Use the Beautifulsoop library to extract references from the XML file
f. Store the titles in the form of a dictionary with the key being the Titles of the pdf (extracted previously) and the value being the extracted References. Let's call it **ref_data_structure** (this will be used for question 3).

Note: Now we do not need to access the pdfs again as we have obtained all the necessary information from them.

Question1:

1. Preprocess Titles obtained from D1

```
]: tile_1 = []
   for i in tile:
       s = preprocess(i) # preprocessing title from d1
       tile_1.append(s)
```

2. Preprocess Titles obtained from D2 in the same way

```
|: # Preproccesing the d2 title list
   key_title_1 = []
   for i in key_title:
       s = preprocess(i)
       key_title_1.append(s)
```

```
|: key_title_1 # printing the title list from d2 after preprocess
```

```
|: ['geometric complexity theory v deciding nonvanishing generalized littlewoodrichardson coefficient dedicated sri ram
   a',
    'recommending related papers based digital library access records',
    'rate bounds mimo relay channels using message splitting',
    '',
    'finite element model updating using response surface method',
    'crystallization large wireless networks',
    'elementary transformation analysis arrayol',
    'freebsd mandatory access control usage implementing enterprise security policies',
    'complexity determining existence hamiltonian cycle 3',
    'pspace bounds rank1 modal logics',
    'building cooperative communications system',
    'kinematic analysis new parallel machine tool orthoglide',
```

3. Some titles from D1 are directly matched to D2. When a match happens we extract corresponding references from the file at runtime and then create a data structure with the title from D1 being the key and references being the value.

```
# intersection of d1 and d2----
res = set()
res = set(key_title_1) & set(tile_1) # Taking out the intersection between d1 & d2 list to take out the files which are directl
print(len(res))
print(list(key_title_1))
print(list(title_1))
```

```
55
['geometric complexity theory v deciding nonvanishing generalized littlewoodrichardson coefficient dedicated sri ramakrishn
a', 'recommending related papers based digital library access records', 'rate bounds mimo relay channels using message splitt
ing', '', 'finite element model updating using response surface method', 'crystallization large wireless networks', 'elementa
ry transformation analysis arrayol', 'freebsd mandatory access control usage implementing enterprise security policies', 'com
plexity determining existence hamiltonian cycle 3', 'pspace bounds rank1 modal logics', 'building cooperative communications
system', 'kinematic analysis new parallel machine tool orthoglide', 'parsimony principles software components metalanguages',
'unfolding convex polyhedra via quasigeodesics', 'characterising web site link structure', 'homogeneous temporal activity pat
terns large online communication space', 'paper 34ck2005revised manuscript mmt ck05 special issue exhaustive study workspace
topologies 3r orthogonal manipulators geometric simplifications', 'estimating random variables random sparse observations',
```

4. And for the remaining titles, we have performed partial matching using cosine similarity with 80% matching.
    a. Since cosine similarity converter every title to a vector and hence it is computationally very expensive.

b. To solve this problem, we have batched the titles from D1 into a set of 10000 titles in one batch and then matched each batch with D2.

```
: # As FILE IS TOO HEAVY FOR COMPUTATION SO BATCHING IT INTO 15 SETS.....
t1 = tile_1[:10000]
t2 = tile_1[10000:20000]
t3 = tile_1[20000:30000]
t4=  tile_1[30000:40000]
t5=tile_1[40000:50000]
t6=tile_1[50000:60000]
t7=tile_1[60000:70000]
t8=tile_1[70000:80000]
t9=tile_1[80000:90000]
t10=tile_1[90000:100000]
t11=tile_1[100000:110000]
t12=tile_1[110000:120000]
t13=tile_1[120000:130000]
t14=tile_1[130000:140000]
t15=tile_1[140000:148929]
```

c. From the matching obtained from the previous step we have then again applied cosine similarity to obtain the best possible matching.

```
print(tf)
tfidf_vectorizer = TfidfVectorizer()
sparse_matrix = tfidf_vectorizer.fit_transform(tf)
doc_term_matrix = sparse_matrix.toarray()

tgt_transform = tfidf_vectorizer.transform([title]).toarray()
tgt_cosine = cosine_similarity(doc_term_matrix,tgt_transform)
#print(tgt_cosine)
index = np.argmax(tgt_cosine)
maxi = tgt_cosine[index][0]
if maxi>0.70:
    #int = key_title_1.index(a[0][0])

    print(maxi)
    print("d1:",title)
    print("***********")
    print("result:",tf[index]) # index of final list
    if key_title[count-1] != '':

        filename = title_[key_title[count-1]]# filename from the dictionary...
        f = os.path.join(dire, filename) # joining the file name with file..
        path = Path(f)
        if path.is_file():


            file1 = open(f,errors = 'ignore')
            soup = file1.read()# reading the file 1..
            bs = BeautifulSoup(soup,"html.parser") # parsing the text in file1..
            #text = bs.find('text').text # finding the text with <Text> in the given file
            ref_list = []
            for reference in bs.find_all('title', type = "main"):
                ref_list.append(reference.get_text())


           # texti = title+text # cocatenating the <Title> and <Text> string..
            file1.close();
            #print(ref_list)
            ref[key_title[count-1]] = ref_list
```

d. While performing the matching, when a match happens we extract corresponding references from the file at runtime and then create a data structure with the title from D1 being the key and references being the value.

```
1.0
d1: evolutionary optimization codebased test compression
**********
result: evolutionary optimization codebased test compression
```

5. Then merge the items obtained from Step3 and Step2 by performing a union operation. This will give us our final data structure.

```
# Final Dictionary----
final_dict
```

```
{'Communication-Efficient Parallel Belief Propagation for Latent Dirichlet Allocation': ['Distributed algorithms for topic mo
dels',
  'Online inference of topics with latent dirichlet allocation',
  'Using variational inference and mapreduce to scale topic modeling',
  'Online learning for latent dirichlet allocation',
  'Latent dirichlet allocation',
  'Learning topic models by belief propagation',
  "Power laws, pareto distributions and zipf's law",
  'Plda: Parallel latent dirichlet allocation for large-scale applications',
  'Finding scientific topics',
  'Human behavior and the principle of least effort',
  'Distributed algorithms for topic models',
  'Plda+: Parallel latent dirichlet allocation with data placement and pipeline processing'],
 'Problem Complexity in Parallel Problem Solving': ['Some Effects of Certain Communication Patterns on Group Performance',
  'The Impact of Certain Communication Nets upon Organization and Performance in Task-oriented Groups',
  'The Effects of Changes in Communication Networks on the Behaviors of Problem-solving Groups',
  'Some Effects of Unequal Distribution of Information upon Group Performance in Various Communication Nets',
  'Communication Structure, Decision Structure and Group Performance',
  'Some Effects of Organization Structure on Group Effectiveness',
```

Question2:

Plot the dictionary obtained from question 1 and obtain the network graph

1. We have taken the final data structure from "filename_final_dict.txt" and entered into l1
2. Import networkx and matplotlib
3. for every refrence that exists for a key, add an edge to the graph
4. plot the graph

```
l1 = {"Communication-Efficient Parallel Belief Propagation for Latent Dirichlet Allocation": ["Distributed algorithms fo
```

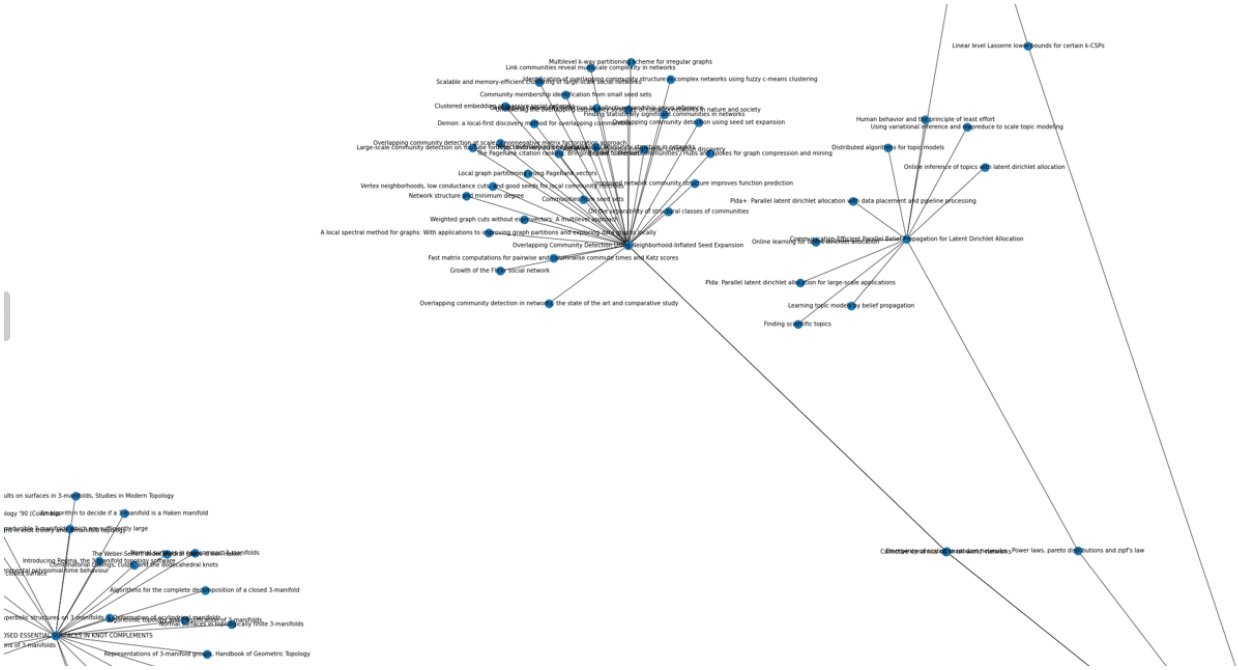
```
#importing libraries
import networkx as nx
import matplotlib.pyplot as plt

g = nx.Graph()
```

```
#for every refrence that exists for a key, add an edge to the graph
for i, j in l1.items():
    if len(j)!=0:        #if there are refrences only then move ahead
        for k in j:
            g.add_edge(i,k)
```

```
fig = plt.figure(1, figsize=(400, 400), dpi=60)
nx.draw(g, with_labels = True, font_weight='normal')
```

Outcome

Lets zoom in



Question 3:

We have used the **ref_data_structure** obtained above.

We then plot this dictionary and obtain a network graph.

===> refer images index and index1 from zip file

```
counte = 0
countue = 0
for i, j in l2.items():
    pos_i = title.index(i)
    val_i = filename[pos_i]
    val_i = val_i.removesuffix('.tei.xml')
#     count +=1
    if len(j)!=0:
        for k in j:
            pos_k = title.index(k)
            val_k = filename[pos_k]
            val_k = val_k.removesuffix('.tei.xml')
            if val_i==val_k:
               counte = counte+1
            else:
                countue = countue+1
                g.add_edge(val_i,val_k)

print('equal ===> '+ str(counte))
print('unequal ===> '+ str(countue))
print('selfloop ===> ' + str(nx.number_of_selfloops(g)))
```

```
equal ===> 84
unequal ===> 1579
selfloop ===> 82
```

```
g.remove_edges_from(nx.selfloop_edges(g))
fig = plt.figure(1, figsize=(200, 100), dpi=60)
nx.draw(g, with_labels = True, font_weight='normal')
```

Outcome:

Lets zoom in: