

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import string
# from unidecode import unidecode # For replacing non-english characters
from nltk.corpus import stopwords
from nltk import word_tokenize
STOPWORDS = set(stopwords.words('english'))
import re
from collections import Counter
import networkx as nx # For graph visualization
```

```
In [2]: pd.set_option('display.max_rows', None)
pd.set_option('display.max_colwidth', None)
```

Step 1: Read and display the script data

```
In [3]: scripts_df = pd.read_csv('../Data/lotr/lotr_scripts.csv')
```

```
In [4]: scripts_df.head(2)
```

```
Out[4]:
```

	Unnamed: 0	char	dialog	movie
0	0	DEAGOL	Oh Smeagol I've got one! , I've got a fish Smeagol, Smeagol!	The Return of the King
1	1	SMEAGOL	Pull it in! Go on, go on, go on, pull it in!	The Return of the King

```
In [5]: scripts_df.shape
```

```
Out[5]: (2390, 4)
```

Step 2: We will start with editing scripts_df

```
In [6]: scripts_df.drop('Unnamed: 0', inplace=True, axis=1) #Remove dummy first column
```

```
In [7]: scripts_df.describe() # Describe the dataframe
```

```
Out[7]:
```

	char	dialog	movie
count	2390	2389	2390
unique	118	2325	3
top	FRODO	DEATH!	The Two Towers
freq	225	6	1010

```
In [8]: scripts_df.dtypes # Look at the types. We notice that they are all  
object so need to convert to string
```

```
Out[8]: char      object  
dialog    object  
movie     object  
dtype: object
```

```
In [9]: # Utility function to get character classes  
def get_char_classes(df, field):  
    print(f"We have {len(df[field].unique())} characters as follows  
for Lord of The Rings:\n {df[field].unique()}")
```

```
In [10]: get_char_classes(scripts_df, 'char')
```

We have 118 characters as follows for Lord of The Rings:

```
['DEAGOL' 'SMEAGOL' '(GOLLUM' 'FRODO' 'MERRY' 'GIMLI' 'GOLLUM' 'SAM'
,
'GANDALF' 'ARAGORN' 'PIPPIN' 'HOBBIT' 'ROSIE' 'BILBO' 'TREEBEARD'
'SARUMAN' 'THEODEN' 'GALADRIL' 'ELROND' 'GRIMA' 'FRODO VOICE OVER'
'WITCH KING' 'EOWYN' 'FARAMIR' 'ORC' '\xa0GANDALF' 'SOLDIERS ON GAT
E'
'GOTHMOG' 'GENERAL' 'CAPTAIN' 'SOLDIER' 'MOUTH OF SAURON' 'EOMER' '
ARMY'
'BOSON' 'MERCENARY' 'EOWYN/MERRY' 'DENETHOR' 'ROHIRRIM'
'GALADRIEL VOICEOVER' 'LEGOLAS' 'GALADRIEL' 'KING OF THE DEAD' 'GRI
MBOLD'
'IROLAS' 'ORCS' 'GAMLING' 'MADRIL' 'DAMROD' 'SOLDIERS'
'SOLDIERS IN MINAS TIRITH' 'GANDALF VOICEOVER' 'SOLDIER 1' 'SOLDIER
2'
'WOMAN' 'HALDIR' 'SAM VOICEOVER' 'OLD MAN' 'BOROMIR' 'CROWD' 'ARWEN'
,
'ELROND VOICEOVER' 'ARWEN VOICEOVER' 'ARAGORN ' 'HAMA' 'SHARKU' 'PE
OPLE'
'LADY' 'FREDA' 'MORWEN' 'EYE OF SAURON' 'ROHAN STABLEMAN' 'GORBAG'
'ARGORN' 'GANDALF VOICE OVER' 'BOROMIR ' 'UGLUK' 'SHAGRAT'
'SARUMAN VOICE OVER' 'SARUMAN VOICE OVER ' 'FRODO ' 'URUK-HAI' 'SNA
GA'
'GRISHNAKH' 'MERRY and PIPPIN' 'WILDMAN' 'STRIDER' 'GALADRIEL VOICE
-OVER'
'EOTHAIN' 'ROHAN HORSEMAN' 'SAURON VOICE' 'SAM ' 'FRODO VOICE'
'GALADRIEL VOICE OVER' 'FARMER MAGGOT' 'WHITE WIZARD' 'MERRY AND PI
PPIN'
'GAFFER' 'NOAKES' 'SANDYMAN' 'FIGWIT' 'GENERAL SHOUT' 'GRISHNAK'
'URUK HAI' 'SARUMAN VOICEOVER' 'MRS BRACEGIRDLE' 'BILBO VOICEOVER'
'PROUDFOOT HOBBIT' 'GATEKEEPER' 'GATEKEEPR' 'MAN' 'CHILDREN HOBBITS'
,
'BARLIMAN' 'RING' 'MEN' 'VOICE' 'SAURON' 'GAN DALF']
```

We see several problems:

1. Upercase all characters
2. '(GOLLUM' has punctuation so need to remove that
3. There are some garbage characters trailing and leading
4. Whenever, VOICE OVER is a suffix, that is the same as the actual character speaking for our purposes
5. Some character names are misspelled such as GRISHNAKH is spelled in two different ways and others like STRIDER and ARAGORN are the same person
6. (optional): remove all white-space. This is a last reserve Solution: we will apply each transformation separately and observe the results

```
In [11]: def remove_voice(character):  
    if character == 'VOICE':  
        character = 'NARRATOR'  
    character = character.replace('VOICEOVER', '')  
    character = character.replace('VOICE OVER', '')  
    character = character.replace('VOICE', '')  
    character = character.strip()  
    return character
```

```
In [12]: def fix_spelling(character):  
    if character == 'STRIDER':  
        character = 'ARAGORN'  
    if 'GAN' in character and 'DALF' in character:  
        character = 'GANDALF'  
    if 'SOLDIER' in character:  
        character = 'SOLDIER'  
    if 'GRISHNAK' in character:  
        character = 'GRICKNAK'  
    if 'URUK' in character:  
        character = 'URUKHAI'  
    if 'GATEKEEP' in character:  
        character = 'GATEKEEPER'  
    if 'ORC' in character:  
        character = 'ORC'  
    if 'GALAD' in character:  
        character = 'GALADRIEL'  
    return character
```

```

In [13]: import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet

lemmatizer = WordNetLemmatizer()

def nltk2wn_tag(nltk_tag):
    if nltk_tag.startswith('J'):
        return wordnet.ADJ
    elif nltk_tag.startswith('V'):
        return wordnet.VERB
    elif nltk_tag.startswith('N'):
        return wordnet.NOUN
    elif nltk_tag.startswith('R'):
        return wordnet.ADV
    else:
        return None

def lemmatize_sentence(sentence):
    nltk_tagged = nltk.pos_tag(nltk.word_tokenize(sentence))
    wn_tagged = map(lambda x: (x[0], nltk2wn_tag(x[1])), nltk_tagged)
    res_words = []
    for word, tag in wn_tagged:
        if tag is None:
            res_words.append(word)
        else:
            res_words.append(lemmatizer.lemmatize(word, tag))

    text = [i for i in res_words if i] ### remove empty strings

    return " ".join(text)

```

```

In [14]: def clean_dialog(text):

    REPLACE_BY_SPACE_RE = re.compile('[/(){}\\[\\]\\|@,;]')
    BAD_SYMBOLS_RE = re.compile('[^0-9a-z #+_]')
    STOPWORDS = set(stopwords.words('english'))

    text = text.lower()
    text = REPLACE_BY_SPACE_RE.sub(' ', text)
    text = BAD_SYMBOLS_RE.sub('', text)
    text = [word for word in text.split() if word not in STOPWORDS]
    #text = [i for i in text if i] ### remove empty strings
    text = ' '.join(text)

    return text[:50] #return only the first 200 characters

```

```
In [15]: def clean_df(df, field):
          df = df.astype(str)
          if field == 'char':

              df[field] = df[field].str.strip()
              df[field] = df[field].str.upper()
              #1. Remove punctuation
              df[field] = scripts_df[field].str.replace('{ }'.format(string.punctuation), '')
              #2.Remove trailing and leading garbage characters
              df[field] = df[field].str.strip()
              #3. Remove voice
              df[field] = df[field].apply(remove_voice)
              #4. Fix spelling
              df[field] = df[field].apply(fix_spelling)

          elif field == 'movie':

              df[field] = df[field].str.strip()
              df[field] = df[field].str.upper()

          elif field == 'dialog':

              df[field] = df[field].apply(clean_dialog)

          #         df[field] = df[field].apply(lemmatize_sentence)

          df = df[df[field].apply(lambda x: len(x)>5)]

          return df
```

```
In [16]: scripts_df = clean_df(scripts_df, 'char')
```

In [17]: `get_char_classes(scripts_df, 'char')`

We have 85 characters as follows for Lord of The Rings:

```
[ 'DEAGOL' 'SMEAGOL' 'GOLLUM' 'FRODO' 'MERRY' 'GIMLI' 'SAM' 'GANDALF'
,
'ARAGORN' 'PIPPIN' 'HOBBIT' 'ROSIE' 'BILBO' 'TREEBEARD' 'SARUMAN'
'THEODEN' 'GALADRIEL' 'ELROND' 'GRIMA' 'WITCH KING' 'EOWYN' 'FARAMI
R'
'ORC' 'SOLDIER' 'GOTHMOG' 'GENERAL' 'CAPTAIN' 'MOUTH OF SAURON' 'EO
MER'
'ARMY' 'BOSON' 'MERCENARY' 'EOWYNMERRY' 'DENETHOR' 'ROHIRRIM' 'LEGO
LAS'
'KING OF THE DEAD' 'GRIMBOLD' 'IROLAS' 'GAMLING' 'MADRIL' 'DAMROD'
'WOMAN' 'HALDIR' 'OLD MAN' 'BOROMIR' 'CROWD' 'ARWEN' 'HAMA' 'SHARKU
,
'PEOPLE' 'LADY' 'FREDA' 'MORWEN' 'EYE OF SAURON' 'ROHAN STABLEMAN'
'GORBAG' 'ARGORN' 'UGLUK' 'SHAGRAT' 'URUKHAI' 'SNAGA' 'GRICKNAK'
'MERRY and PIPPIN' 'WILDMAN' 'EOTHAIN' 'ROHAN HORSEMAN' 'SAURON'
'FARMER MAGGOT' 'WHITE WIZARD' 'MERRY AND PIPPIN' 'GAFFER' 'NOAKES'
'SANDYMAN' 'FIGWIT' 'GENERAL SHOUT' 'MRS BRACEGIRDLE' 'PROUDFOOT HO
BBIT'
'GATEKEEPER' 'MAN' 'CHILDREN HOBBITS' 'BARLIMAN' 'RING' 'MEN' 'NARR
ATOR']
```

```
In [18]: scripts_df.char.value_counts()
```



```
Out[18]: FRODO                229
          SAM                  218
          GANDALF              215
          ARAGORN              212
          PIPPIN               163
          MERRY                137
          GOLLUM               134
          GIMLI                116
          THEODEN              110
          FARAMIR              65
          EOWYN                56
          LEGOLAS              55
          SMEAGOL              49
          BILBO                48
          TREEBEARD            46
          DENETHOR             45
          SARUMAN              41
          BOROMIR              41
          SOLDIER              41
          ARWEN                40
          EOMER                36
          ELROND               31
          GRIMA                25
          ORC                  25
          GALADRIEL            19
          GAMLING              15
          GOTHMOG              14
          UGLUK                9
          SHAGRAT              9
          KING OF THE DEAD     8
          WITCH KING           8
          MADRIL               7
          HALDIR               7
          URUKHAI              6
          HAMA                  6
          MOUTH OF SAURON      5
          GRICKNAK             5
          MORWEN               5
          BARLIMAN             4
          GORBAG               3
          SAURON               3
          CROWD                3
          IROLAS               3
          GAFFER               3
          MERRY and PIPPIN     3
          FREDA                3
          GATEKEEPER           3
          HOBBIT               3
          DEAGOL               3
          CHILDREN HOBBITS     3
          WILDMAN              3
          WOMAN                3
          PEOPLE               2
          CAPTAIN              2
          NARRATOR             2
          ROHAN HORSEMAN       2
```

SNAGA	2
ARMY	2
DAMROD	2
ROHIRRIM	2
SANDYMAN	2
ROSIE	2
MERRY AND PIPPIN	2
ARGORN	2
GENERAL	2
EOTHAIN	1
FARMER MAGGOT	1
MERCENARY	1
SHARKU	1
GENERAL SHOUT	1
PROUDFOOT HOBBIT	1
MRS BRACEGIRDLE	1
MEN	1
NOAKES	1
EOWYNMERRY	1
WHITE WIZARD	1
OLD MAN	1
ROHAN STABLEMAN	1
RING	1
GRIMBOLD	1
LADY	1
FIGWIT	1
BOSON	1
MAN	1
EYE OF SAURON	1

Name: char, dtype: int64

```
In [19]: # Extract top 10 chars in each movie
```

```
In [20]: # Extract the top 10 characters by lines
top10_chars_allmovies = scripts_df.char.value_counts().index.tolist()[:10]
```

```
In [21]: top10_chars_allmovies
```

```
Out[21]: ['FRODO',
          'SAM',
          'GANDALF',
          'ARAGORN',
          'PIPPIN',
          'MERRY',
          'GOLLUM',
          'GIMLI',
          'THEODEN',
          'FARAMIR']
```

Based on the character value counts, it appears that Frodo and Sam have the most lines in all the movies and in general, the top 4 characters have the most lines in all the movies

Based on this scripts value counts, most of the lines are unique

```
In [22]: scripts_df.movie.value_counts()
```

```
Out[22]: The Two Towers          1010  
The Return of the King          873  
The Fellowship of the Ring      507  
Name: movie, dtype: int64
```

Based on the movie value counts, the second movie "The Two Towers" has the most dialogue.

```
In [23]: scripts_df.movie.unique()
```

```
Out[23]: array(['The Return of the King ', 'The Two Towers ',  
               'The Fellowship of the Ring '], dtype=object)
```

```
In [24]: scripts_df = clean_df(scripts_df, 'movie')
```

We now start preprocessing the actual dialog data

```
In [48]: scripts_df[['dialog']].head(50)
```

Out [48]:

	dialog
11	murderer
16	gandalf
17	oooohhh
19	aaaahh
21	precious
22	wake wake wake sleepies must go yes must go
23	havent sleep mr frodo
24	ive gone much must getting late
25	isnt isnt midday yet days growing darker
26	come must go time
27	mr frodos something eat
28	time lose silly
31	oh im hungry leastways lembas bread
33	aragorn
34	come days king may blessed
35	day belong one man let us together rebuild world m
36	et erello endoreenna utlien sinome maruvan ar hildi
37	hannon le
38	friends bow noone
39	thus fourth age middle earth began fellowship ring
41	alright
44	oh alright dont much left careful going run go ahe
46	journey home
47	come hobbitises close close mordor safe places hurr
48	good definitely shire longbottom leaf eh
49	uhh huh feel like im back green dragon
50	mmmm green dragon
51	mug ale hand putting feet settle hard days work
52	youve never done hard days work
54	welcome lords isengard
55	young rascals merry hunt youve led us find feastin
57	pick threads old life go heart begin understand go

	dialog
58	mr frodo
59	four years day since weathertop sam never really h
60	back hobbits tale bilbo baggins lord rings frodo b
61	quite theres room little
62	bilbo told part tale would end us must come go tel
64	harbour bilbo elves accorded special honour place
66	sitting field victory enjoying well earned comfort
67	salted pork
68	hobbits
69	orders treebeard whos taken management isengard
72	careful even defeat saruman dangerous
73	well lets head done
74	need alive need talk
76	shall peace shall peace answer burning westfold ch
78	treachery already cost many lives thousands risk c

In [26]: `len(scripts_df.dialog.unique())`

Out[26]: 2326

In [27]: `scripts_df = clean_df(scripts_df, 'dialog')`

In [47]: `scripts_df[['dialog']].head(10)`

Out[47]:

	dialog
11	murderer
16	gandalf
17	oooohhh
19	aaaahh
21	precious
22	wake wake wake sleepies must go yes must go
23	havent sleep mr frodo
24	ive gone much must getting late
25	isnt isnt midday yet days growing darker
26	come must go time

```
In [29]: scripts_df.dialog.isna().any()
```

```
Out[29]: False
```

```
In [30]: scripts_df['corpus_dialog'] = scripts_df.dialog.apply(lambda x: Counter(x.split()))
```

```
In [46]: scripts_df[['corpus_dialog']].head(10)
```

```
Out[46]:
```

	corpus_dialog
11	{'murderer': 1}
16	{'gandalf': 1}
17	{'ooooohhh': 1}
19	{'aaaahh': 1}
21	{'precious': 1}
22	{'wake': 3, 'sleepies': 1, 'must': 2, 'go': 2, 'yes': 1}
23	{'havent': 1, 'sleep': 1, 'mr': 1, 'frodo': 1}
24	{'ive': 1, 'gone': 1, 'much': 1, 'must': 1, 'getting': 1, 'late': 1}
25	{'isnt': 2, 'midday': 1, 'yet': 1, 'days': 1, 'growing': 1, 'darker': 1}
26	{'come': 1, 'must': 1, 'go': 1, 'time': 1}

```
In [32]: scripts_df = scripts_df[scripts_df.char.isin(top10_chars_allmovies)]
```

```
In [33]: scripts_df.to_csv('../Data/lotr/prepped_data.csv', index=False)
```

Visualizations

Step 3: Now we will look at the vocabulary of each character over the course of the movies to see any transformations such as growth/regression

```
In [34]: fellowship_df = scripts_df[scripts_df.movie=='THE FELLOWSHIP OF THE RING']
two_towers_df = scripts_df[scripts_df.movie=='THE TWO TOWERS']
return_king_df = scripts_df[scripts_df.movie=='THE RETURN OF THE KING']
```

Get top 10 chars for each movies separately and then find the intersection

```
In [35]: top10_chars_1 = fellowship_df.char.value_counts().index.tolist()[1:10]
top10_chars_2 = two_towers_df.char.value_counts().index.tolist()[1:10]
top10_chars_3 = return_king_df.char.value_counts().index.tolist()[1:10]
```

```
In [ ]:
```

```
fellowship_df.groupby('char')['dialog'].count().reset_index(name='obs').sort_values(['obs'],
ascending=False).head(10)
```

```
In [36]: two_towers_df.groupby('char')['dialog'].count().reset_index(name='obs').sort_values(['obs'], ascending=False).head(10)
```

```
Out [36]:
```

	char	obs
0	ARAGORN	85
8	SAM	81
5	GOLLUM	68
2	FRODO	66
9	THEODEN	59
4	GIMLI	51
3	GANDALF	46
6	MERRY	44
7	PIPPIN	42
1	FARAMIR	34


```
In [37]: return_king_df.groupby('char').size().reset_index(name='obs').sort_
values(['obs'], ascending=False).head(10)
```

Out[37]:

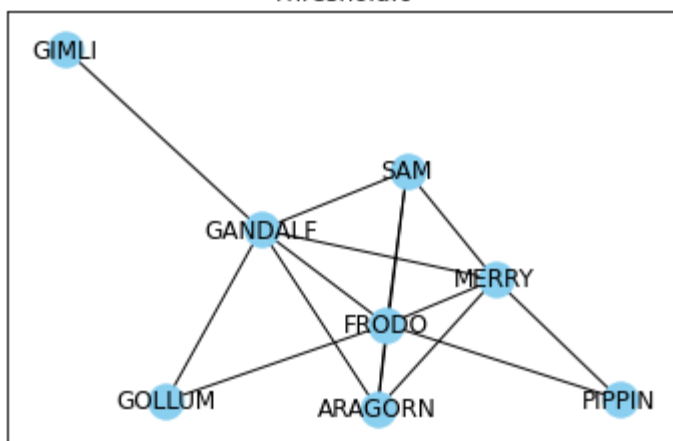
	char	obs
3	GANDALF	84
8	SAM	77
2	FRODO	58
7	PIPPIN	52
0	ARAGORN	49
5	GOLLUM	47
9	THEODEN	39
6	MERRY	33
4	GIMLI	30
1	FARAMIR	22

```
In [38]: def consolidate_char_vocab(df, character):
char_vocabularies = df[df.char == character].corpus_dialog
#Dummy dictionary for plotting character growth
if len(char_vocabularies) == 0:
    my_dict = {'NA1':0, 'NA2':0, 'NA3':0, 'NA4':0, 'NA5':0}
    return my_dict
list_of_dicts = char_vocabularies.tolist()
master_dict = list_of_dicts[0]
for dictionary in list_of_dicts[1:]:
    master_dict = master_dict + dictionary
return master_dict
```

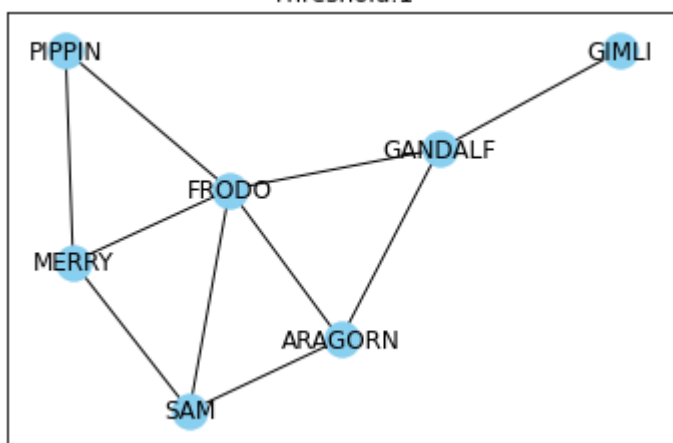
```
In [39]: def get_relationships(df, char_list, thresh):  
    # Get char vocabularies for a specific movie  
    dict_of_char_vocabs = {}  
    for char in char_list:  
        dict_of_char_vocabs[char] = consolidate_char_vocab(df, char)  
    r)  
  
    # Create visual  
  
    edge_list = []  
    visited = set()  
    for char in dict_of_char_vocabs:  
        vocab = dict_of_char_vocabs[char]  
  
        for word in vocab:  
            if word.upper() in dict_of_char_vocabs and vocab[word]  
> thresh:  
                visited.add(word)  
                edge_list.append((char, word.upper()))  
  
    G = nx.Graph()  
    G.add_edges_from(edge_list)  
  
    nx.draw_networkx(G=G, node_size=300, node_color='#89CFF0')  
    plt.title(f"Threshold:{i}")  
  
    plt.show()
```

```
In [40]: for i in range(34):  
         get_relationships(fellowship_df, fellowship_df.char.unique().to  
         list(), i)
```

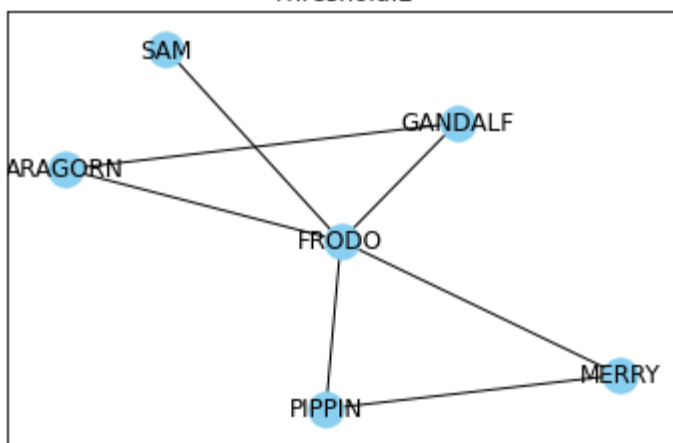
Threshold:0



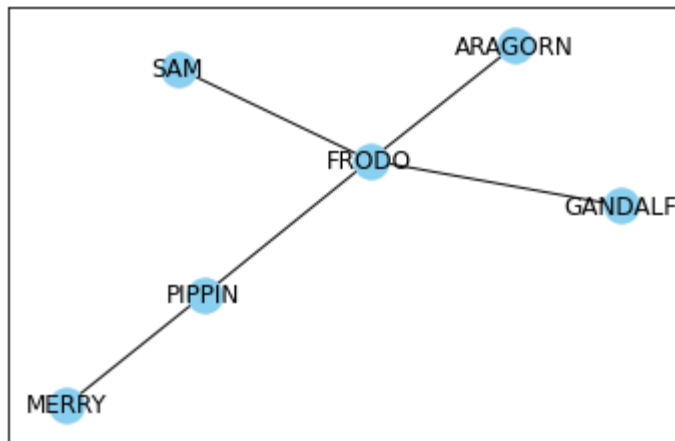
Threshold:1



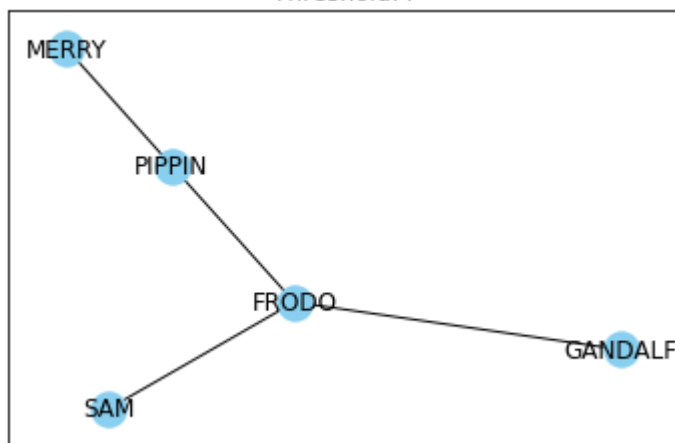
Threshold:2



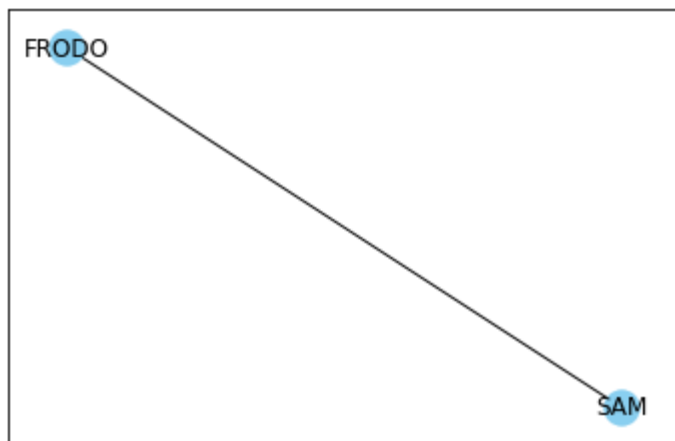
Threshold:3



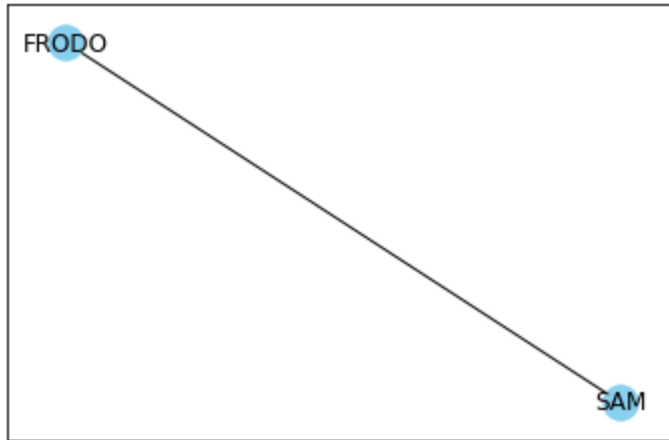
Threshold:4



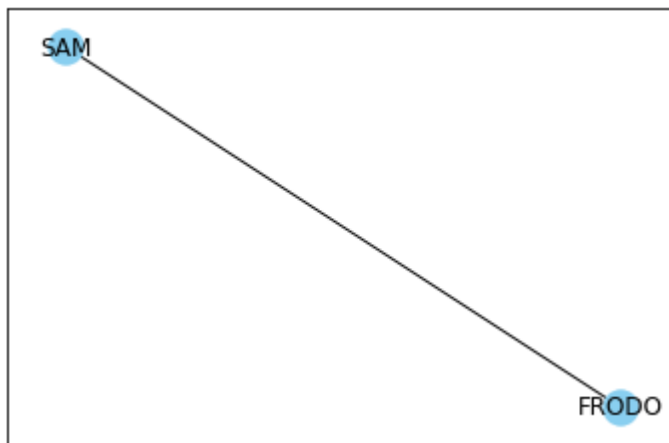
Threshold:5



Threshold:6



Threshold:7



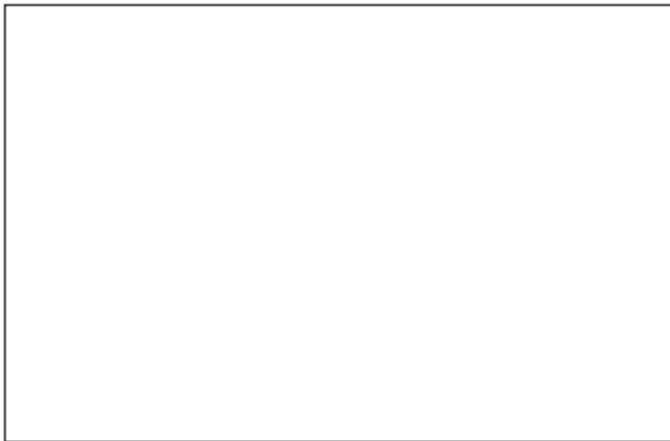
Threshold:8



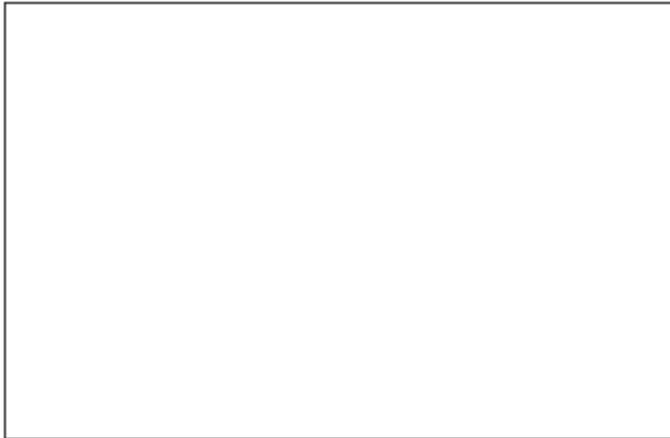
Threshold:9



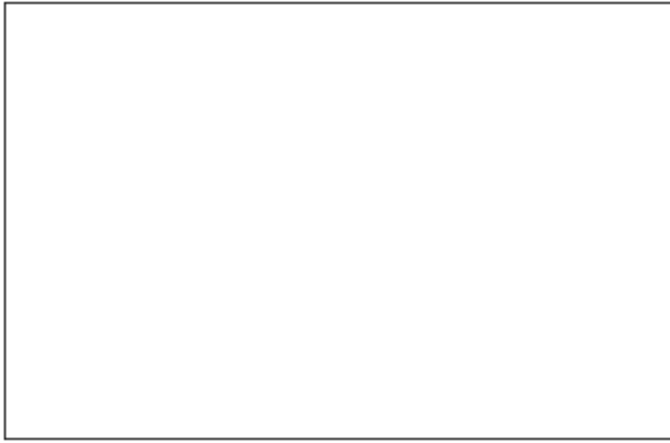
Threshold:10



Threshold:11



Threshold:12



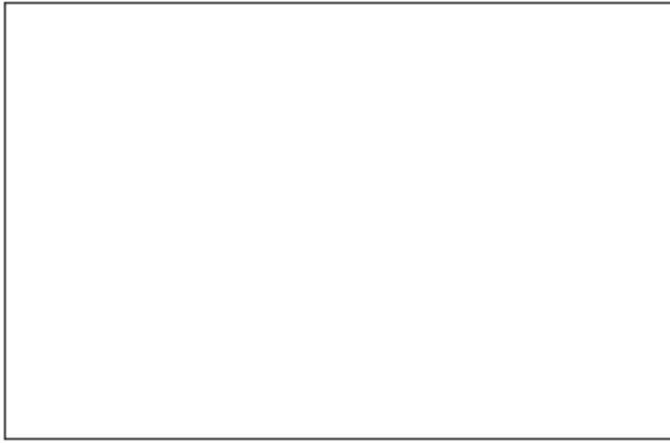
Threshold:13



Threshold:14



Threshold:15



Threshold:16



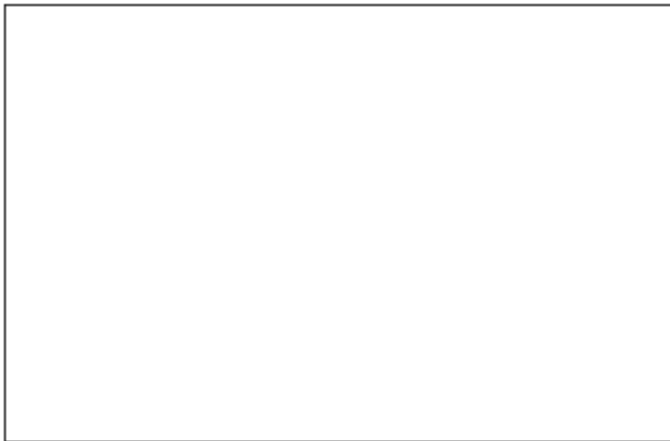
Threshold:17



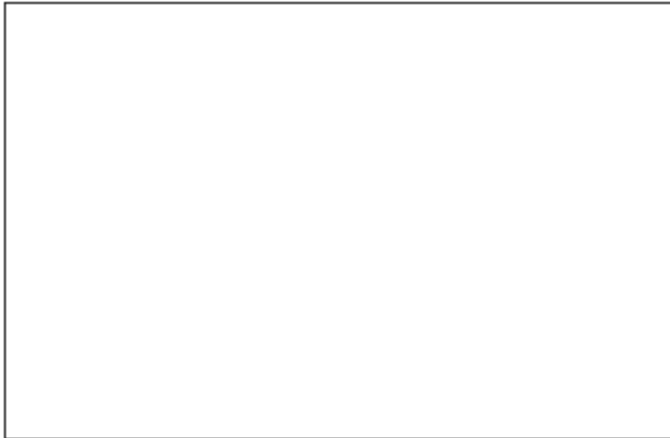
Threshold:18



Threshold:19



Threshold:20



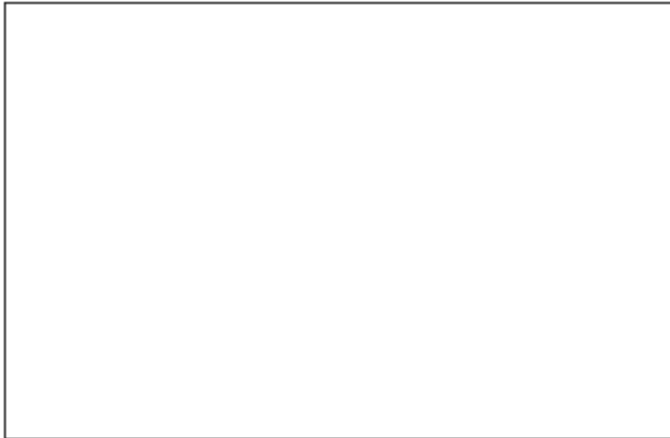
Threshold:21



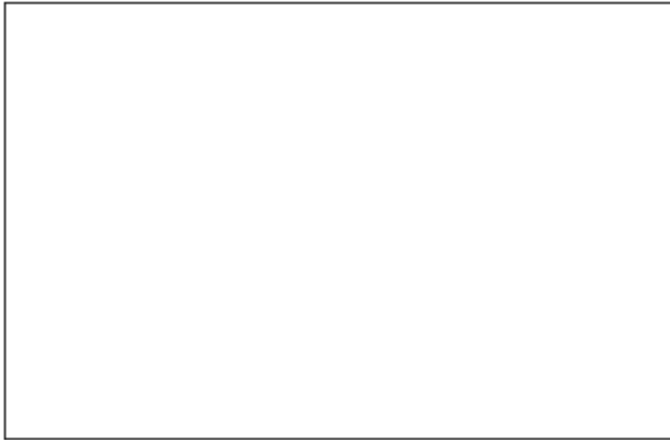
Threshold:22



Threshold:23



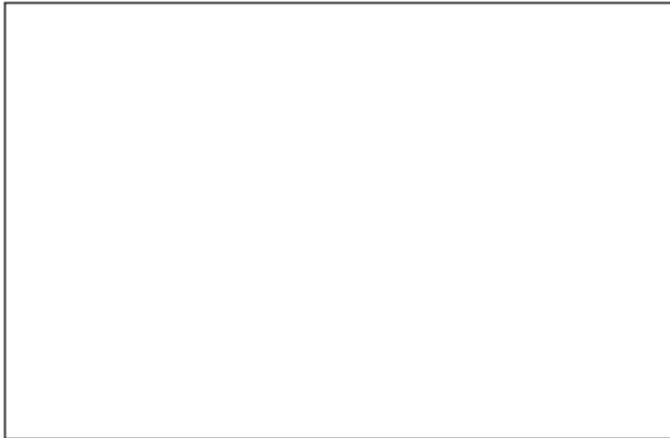
Threshold:24



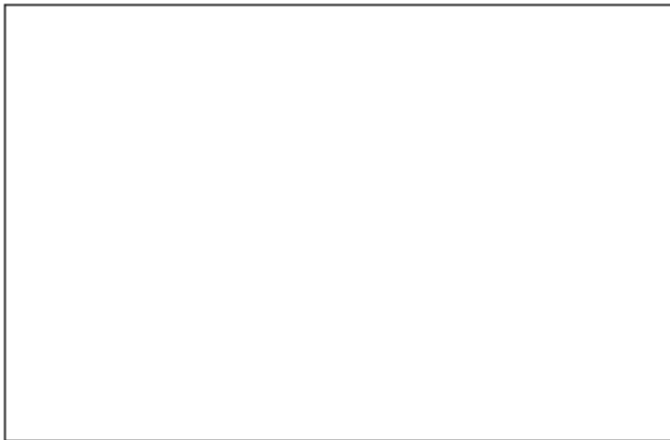
Threshold:25



Threshold:26



Threshold:27



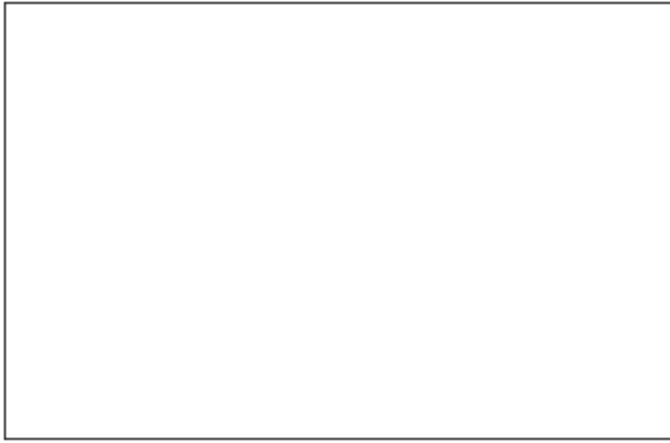
Threshold:28



Threshold:29



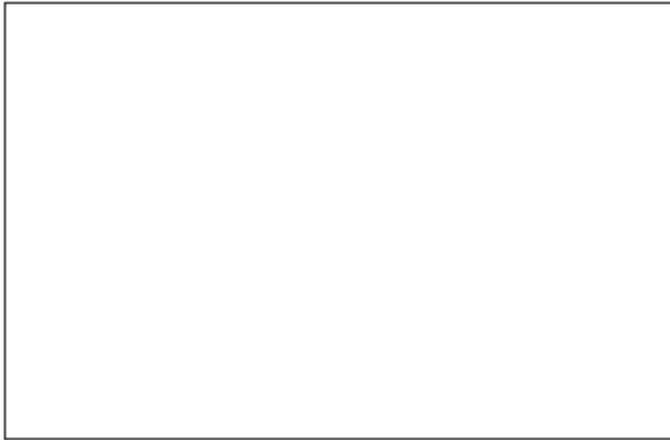
Threshold:30



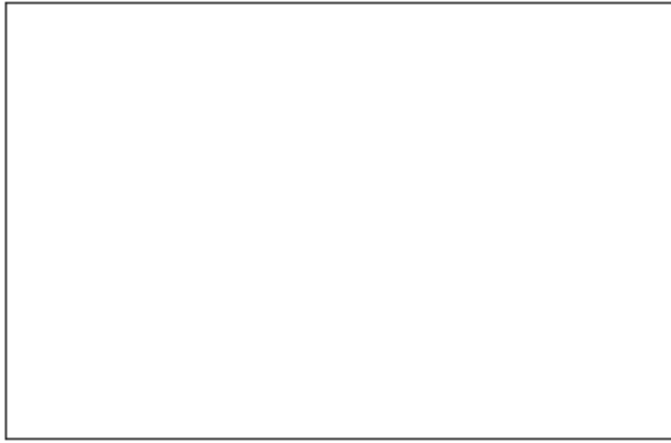
Threshold:31



Threshold:32

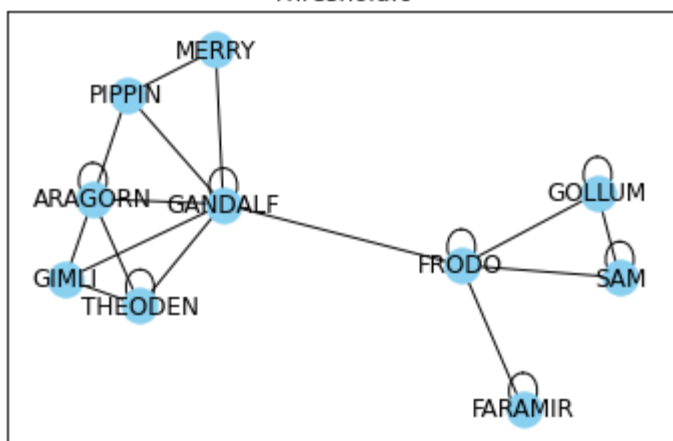


Threshold:33

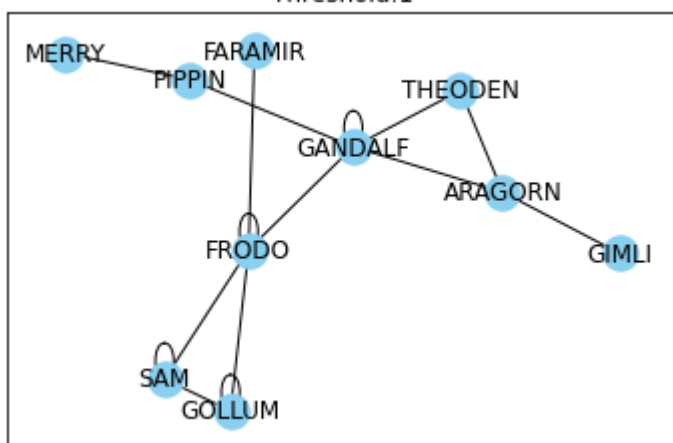


```
In [41]: for i in range(34):  
         get_relationships(two_towers_df, two_towers_df.char.unique().to  
         list(), i)
```

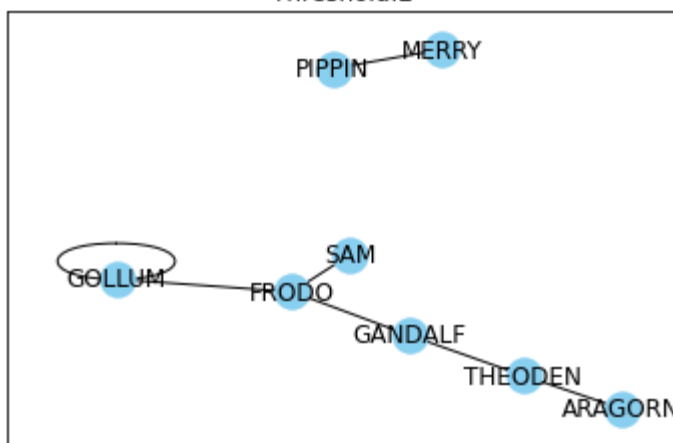

Threshold:0



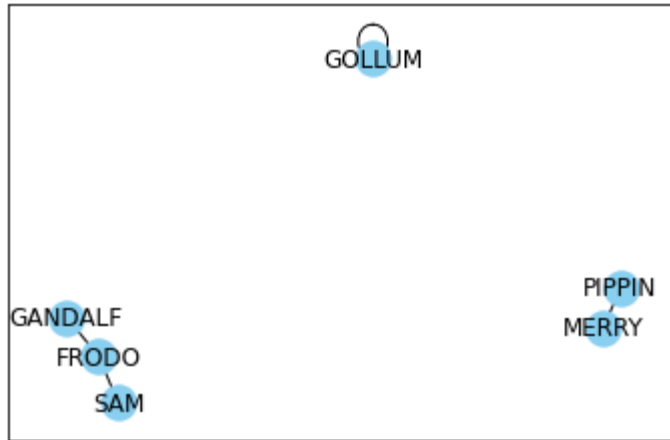
Threshold:1



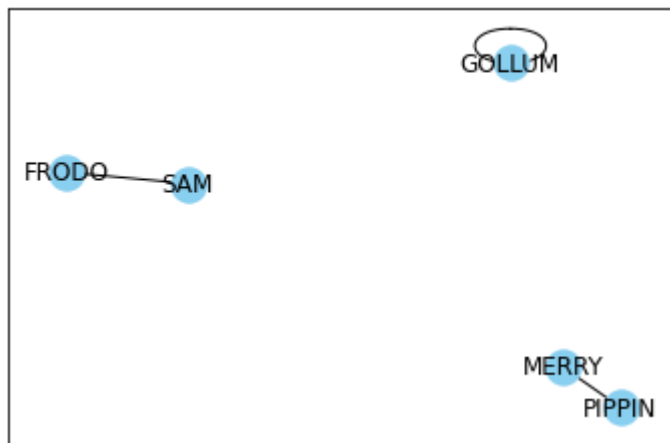
Threshold:2



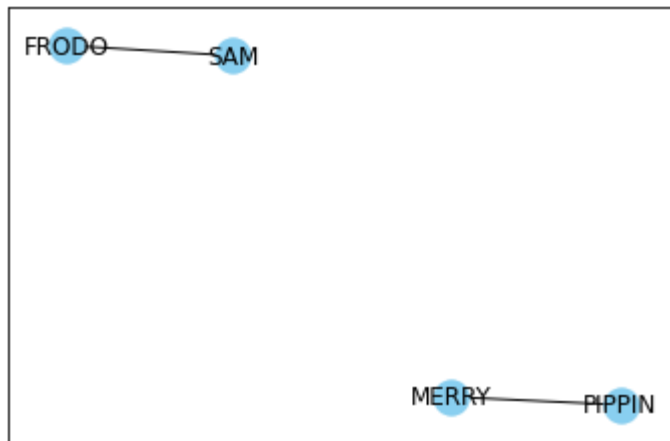
Threshold:3



Threshold:4



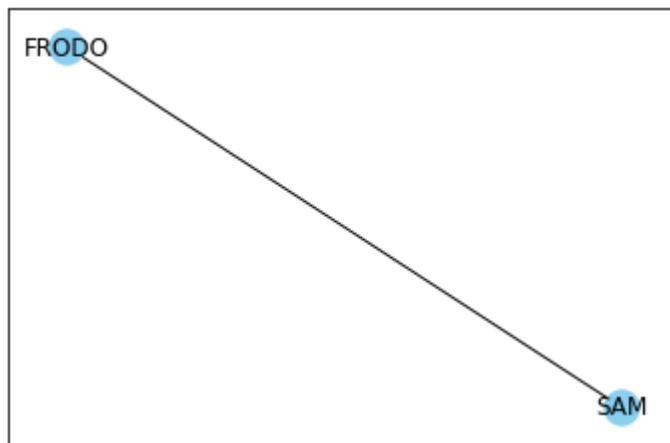
Threshold:5



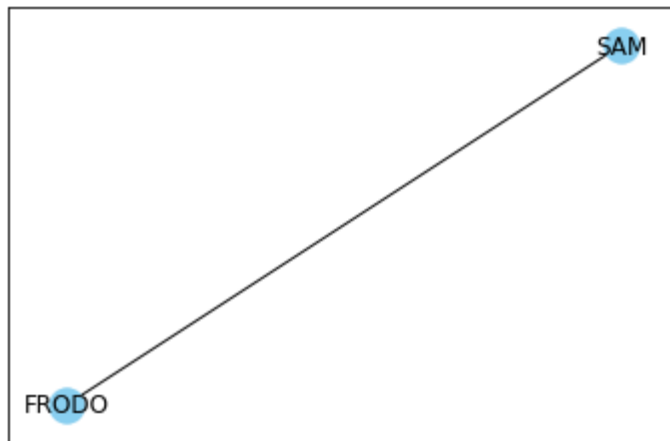
Threshold:6



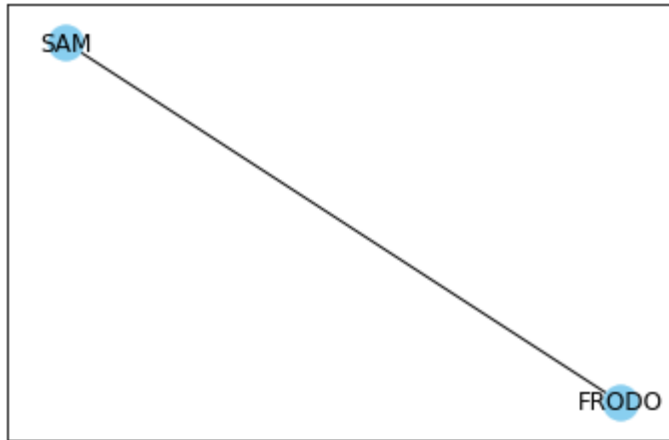
Threshold:7



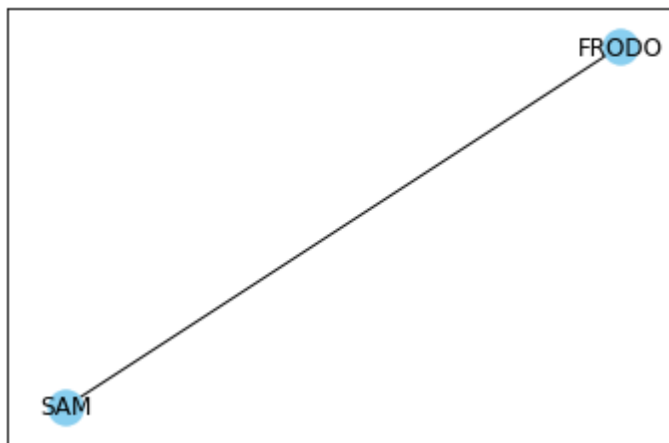
Threshold:8



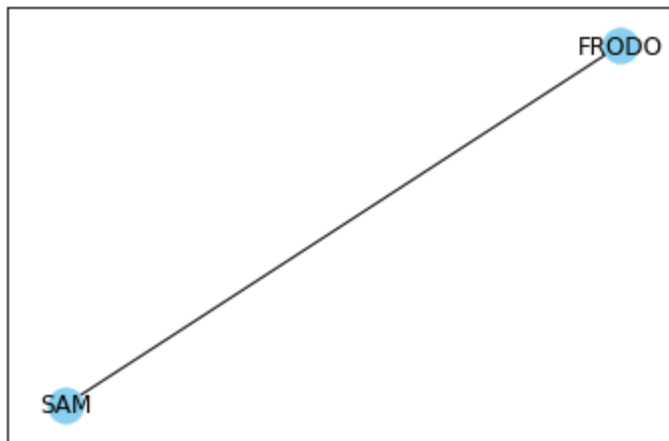
Threshold:9



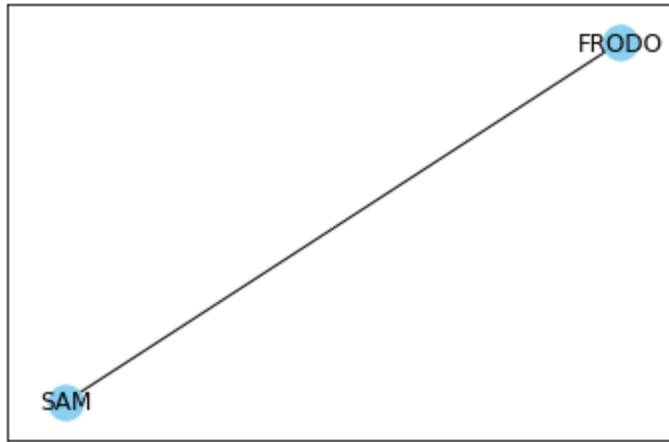
Threshold:10



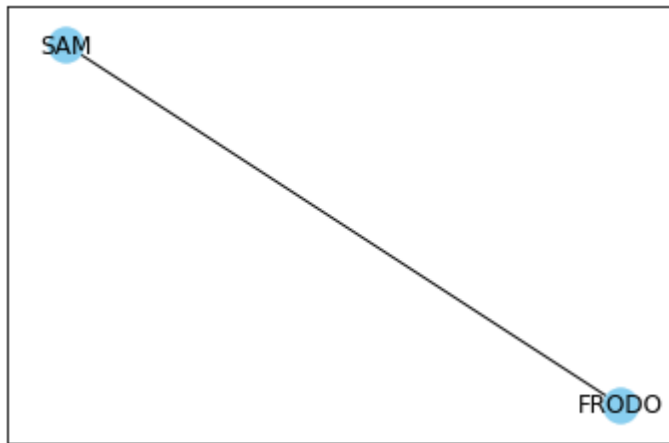
Threshold:11



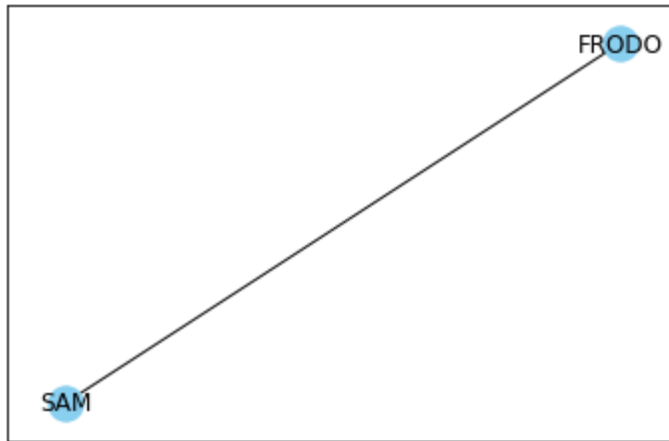
Threshold:12



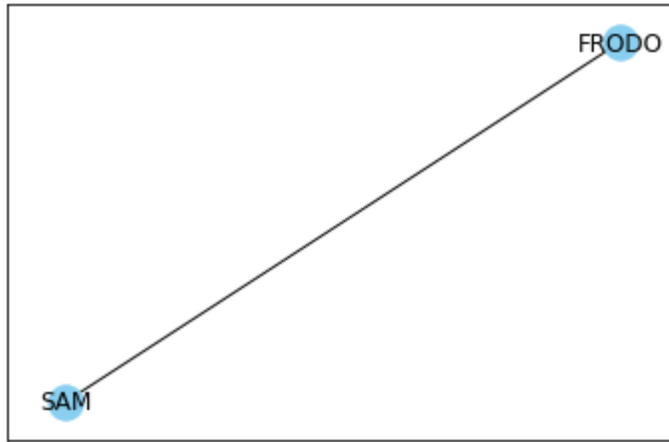
Threshold:13



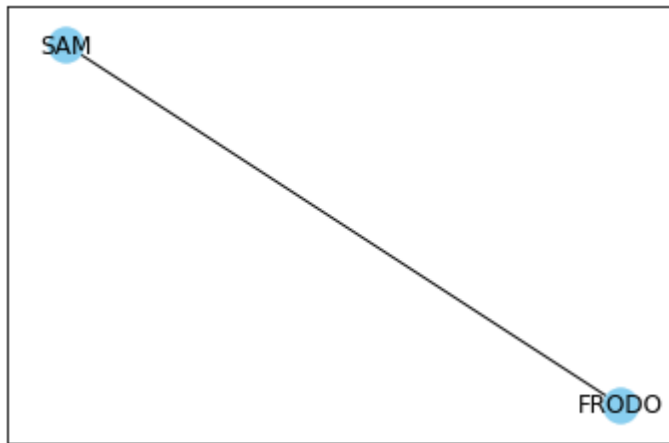
Threshold:14



Threshold:15



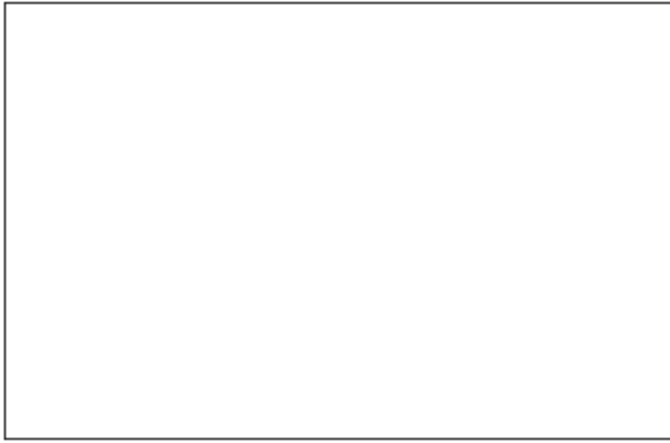
Threshold:16



Threshold:17



Threshold:18



Threshold:19



Threshold:20



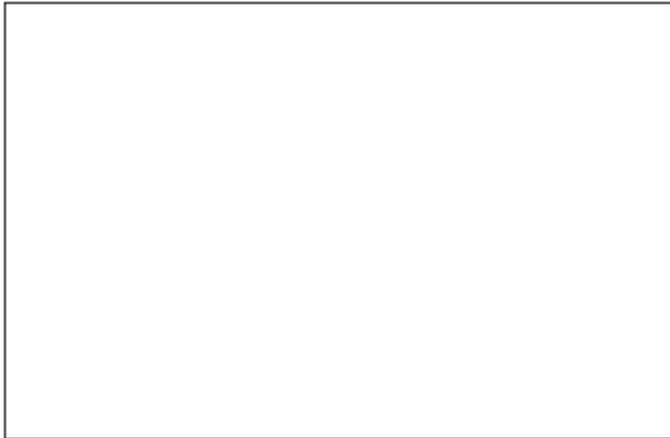
Threshold:21



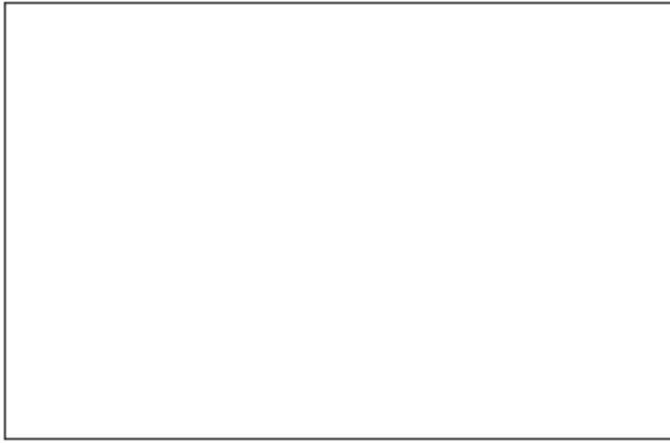
Threshold:22



Threshold:23



Threshold:24



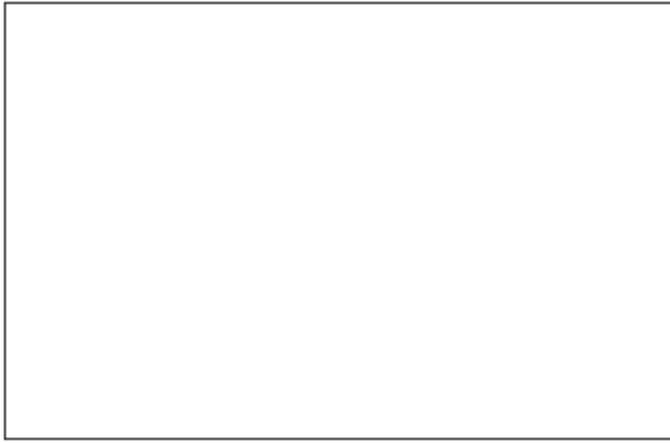
Threshold:25



Threshold:26



Threshold:27



Threshold:28



Threshold:29



Threshold:30



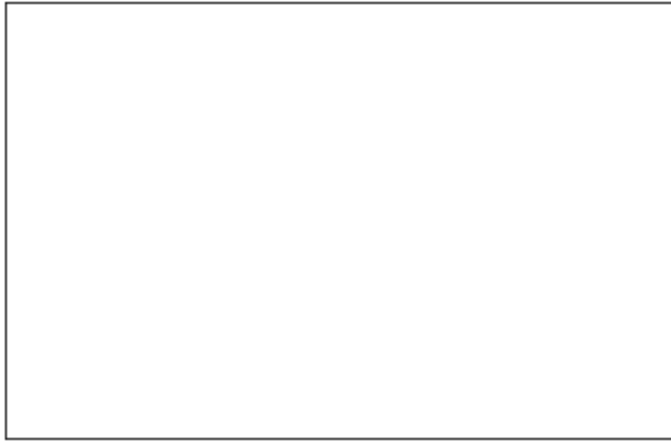
Threshold:31



Threshold:32

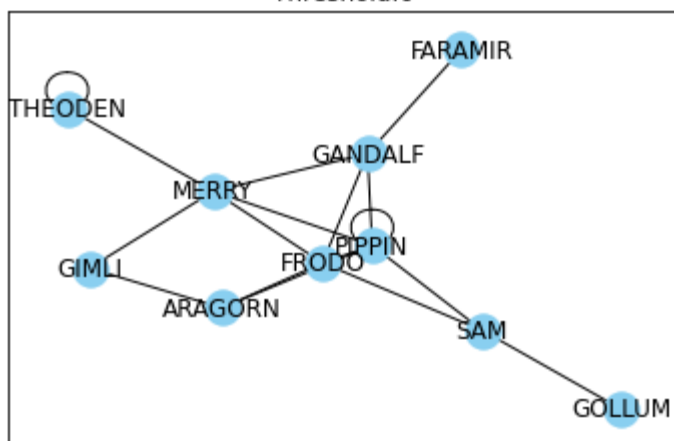


Threshold:33

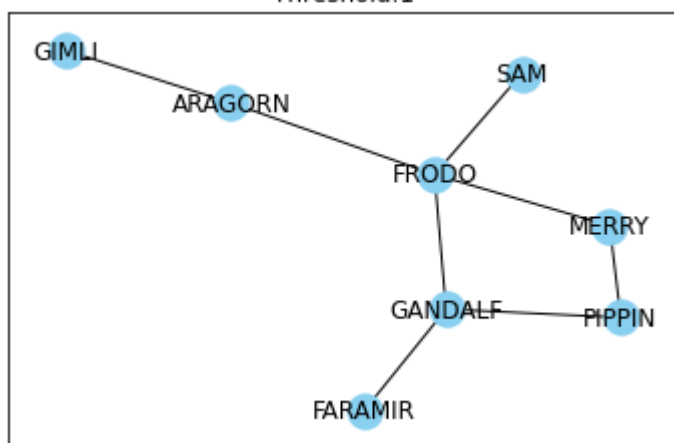


```
In [42]: for i in range(34):  
         get_relationships(return_king_df, return_king_df.char.unique().  
         tolist(), i)
```

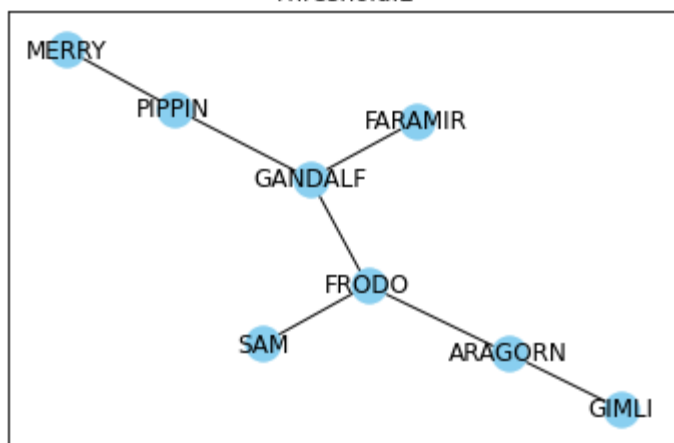
Threshold:0



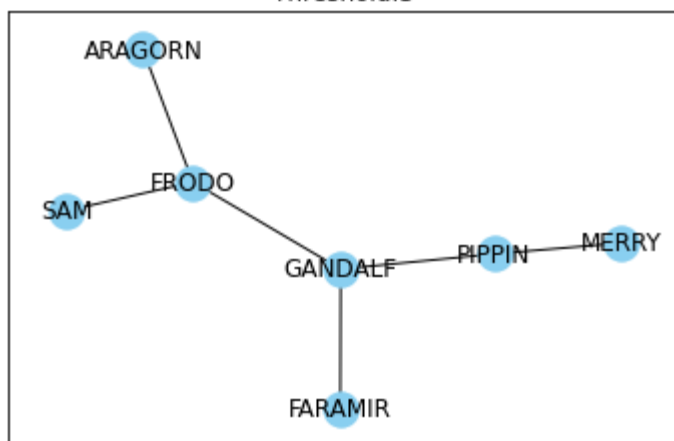
Threshold:1



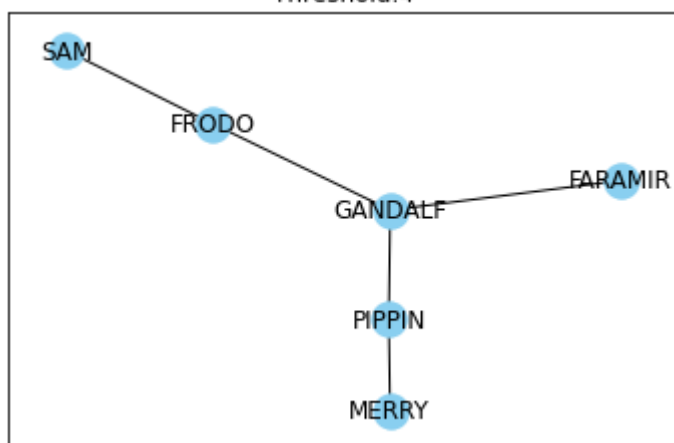
Threshold:2



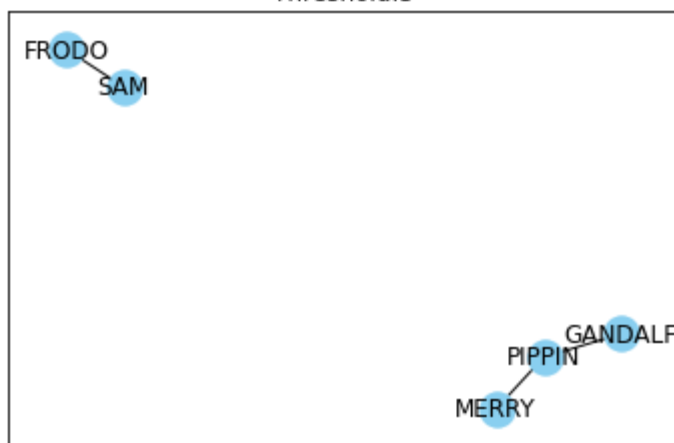
Threshold:3



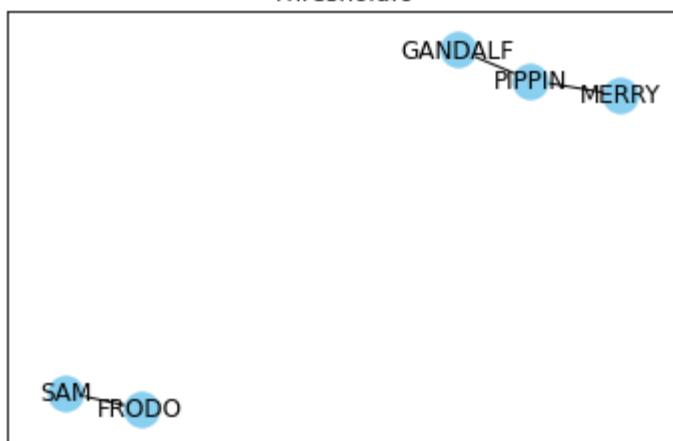
Threshold:4



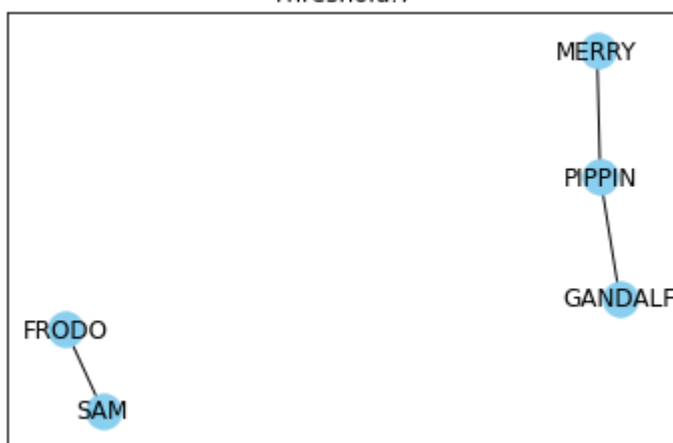
Threshold:5



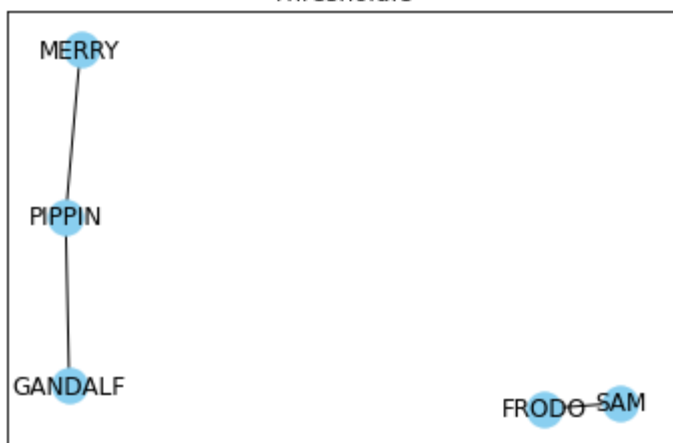
Threshold:6



Threshold:7



Threshold:8



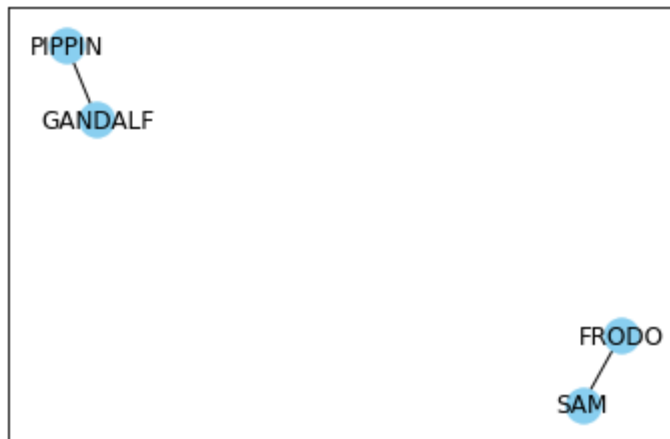
Threshold:9



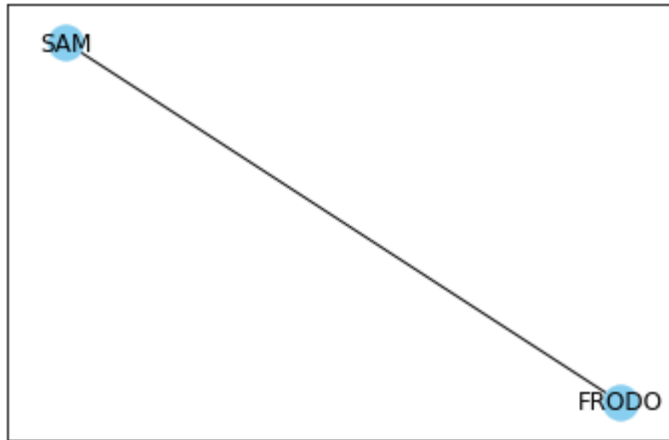
Threshold:10



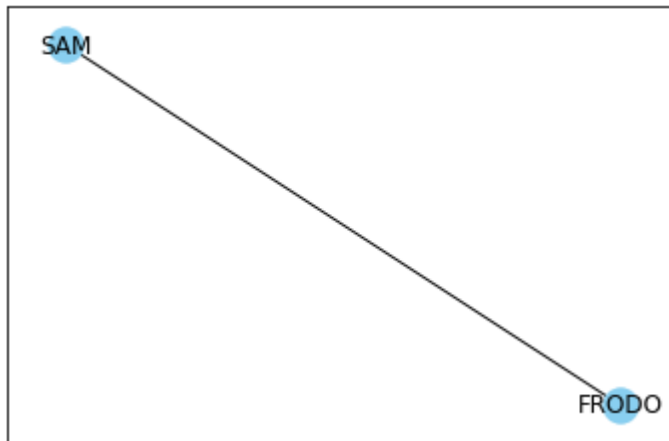
Threshold:11



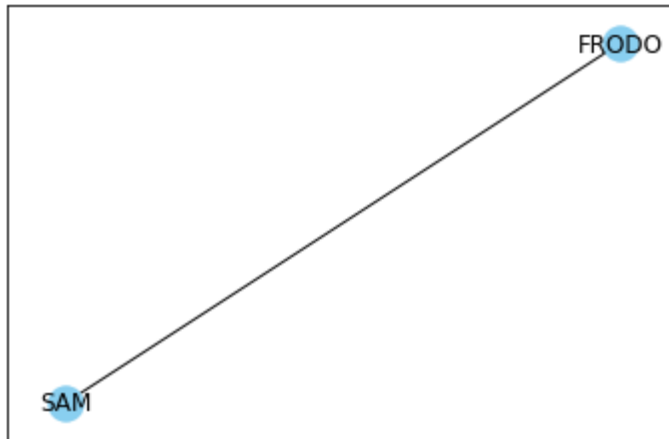
Threshold:12



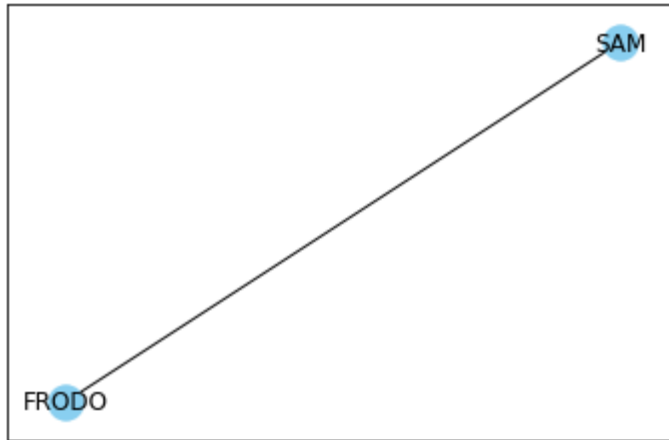
Threshold:13



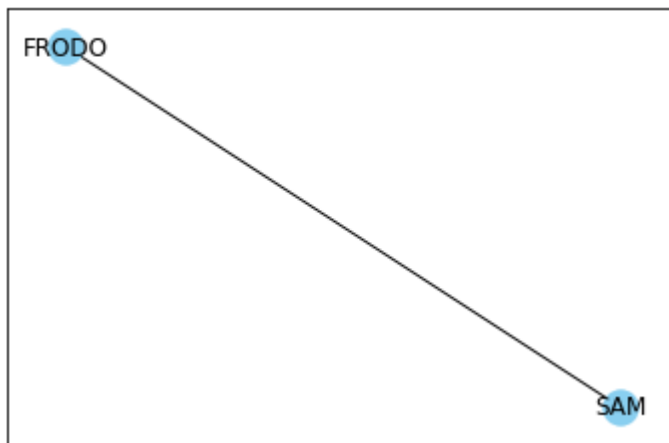
Threshold:14



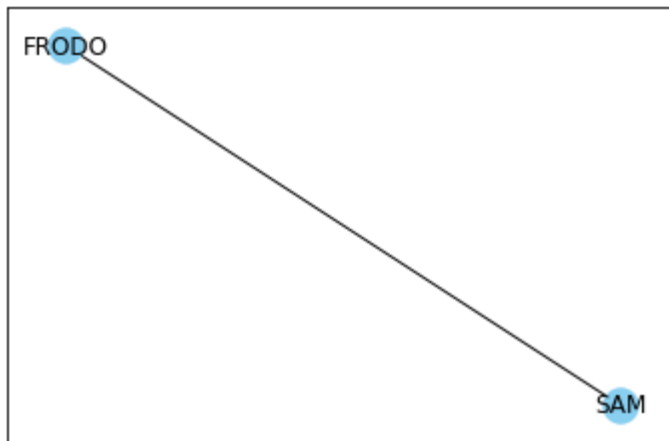
Threshold:15



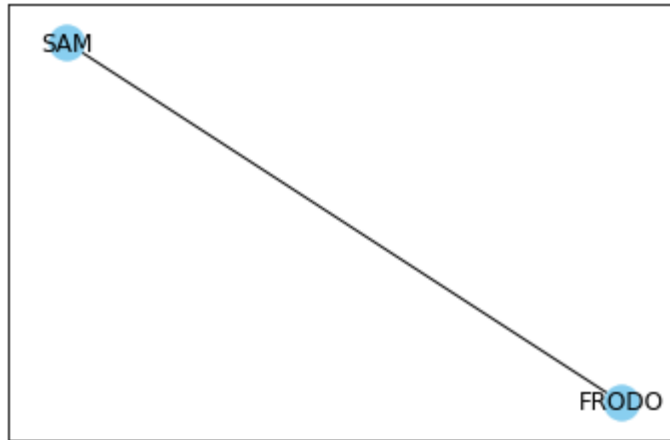
Threshold:16



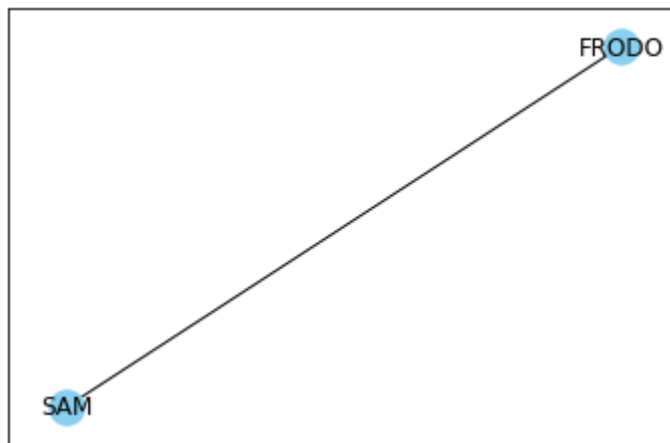
Threshold:17



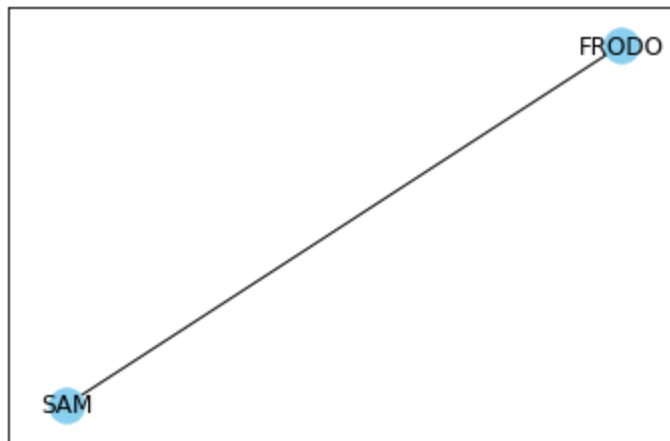
Threshold:18



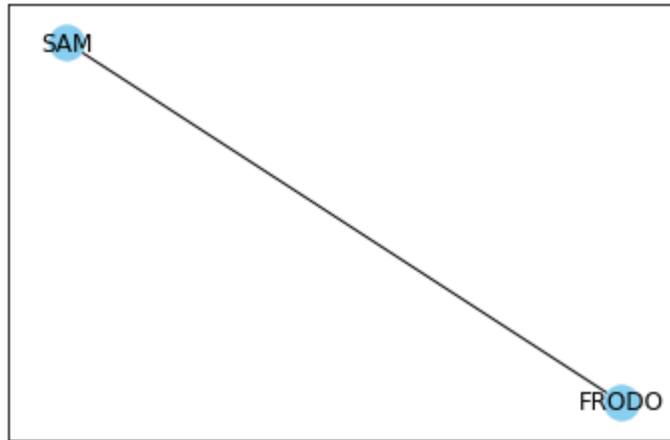
Threshold:19



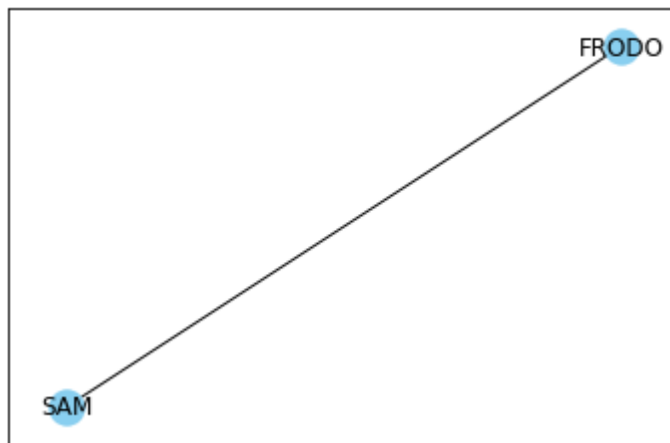
Threshold:20



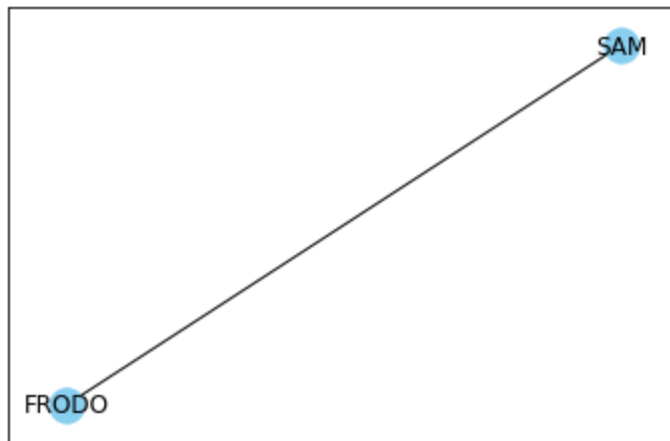
Threshold:21



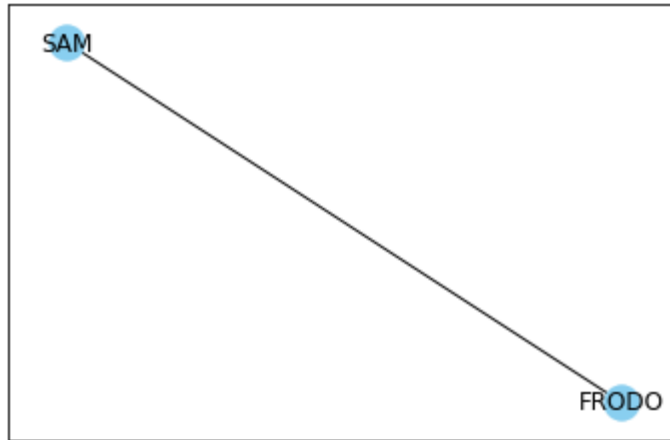
Threshold:22



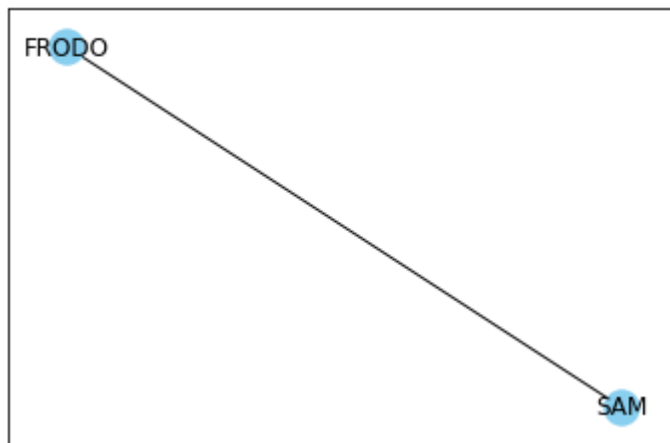
Threshold:23



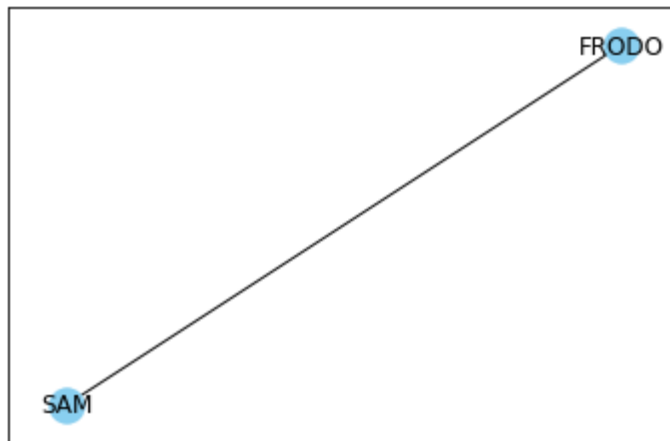
Threshold:24



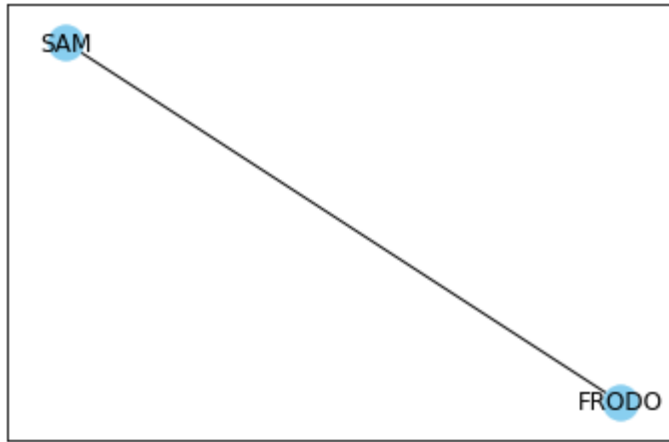
Threshold:25



Threshold:26



Threshold:27



Threshold:28



Threshold:29



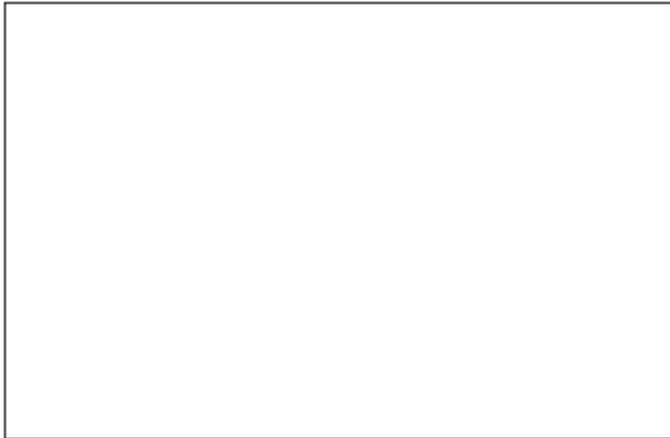
Threshold:30



Threshold:31



Threshold:32



Threshold:33



Now let us try to see if there was any character growth for the top 10 characters

```
In [43]: top10_chars_allmovies
```

```
Out[43]: ['FRODO',  
          'SAM',  
          'GANDALF',  
          'ARAGORN',  
          'PIPPIN',  
          'MERRY',  
          'GOLLUM',  
          'GIMLI',  
          'THEODEN',  
          'FARAMIR']
```

```

In [44]: def get_char_growth(df_list, char, i):

    colors = ['green', 'blue', 'teal', 'indigo', 'limegreen',
              'mediumblue', 'fuchsia', 'cyan', 'coral', 'seagreen']

    # Get char vocabulary

    char_vocab_1 = consolidate_char_vocab(df_list[0], char)
    char_vocab_2 = consolidate_char_vocab(df_list[1], char)
    char_vocab_3 = consolidate_char_vocab(df_list[2], char)

    # Get top 5 words for character in each movie
    top5_words_1 = sorted(char_vocab_1, key=char_vocab_1.get, reverse=True)[:5]
    top5_words_2 = sorted(char_vocab_2, key=char_vocab_2.get, reverse=True)[:5]
    top5_words_3 = sorted(char_vocab_3, key=char_vocab_3.get, reverse=True)[:5]

    # Get subset dictionary for the character based on top 5 words only
    subset_1 = {key: char_vocab_1[key] for key in top5_words_1}
    subset_2 = {key: char_vocab_2[key] for key in top5_words_2}
    subset_3 = {key: char_vocab_3[key] for key in top5_words_3}

    names_1 = list(subset_1.keys())
    names_2 = list(subset_2.keys())
    names_3 = list(subset_3.keys())

    values_1 = list(subset_1.values())
    values_2 = list(subset_2.values())
    values_3 = list(subset_3.values())

    figure, (ax1, ax2, ax3) = plt.subplots(3)

    # For first movie
    ax1.bar(range(len(subset_1)), values_1, tick_label=names_1, color=colors[i])
    ax1.set_title(f"{char} in: {df_list[0].movie.unique()[0]}")

    # For second movie
    ax2.bar(range(len(subset_2)), values_2, tick_label=names_2, color=colors[i])
    ax2.set_title(f"{char} in: {df_list[1].movie.unique()[0]}")

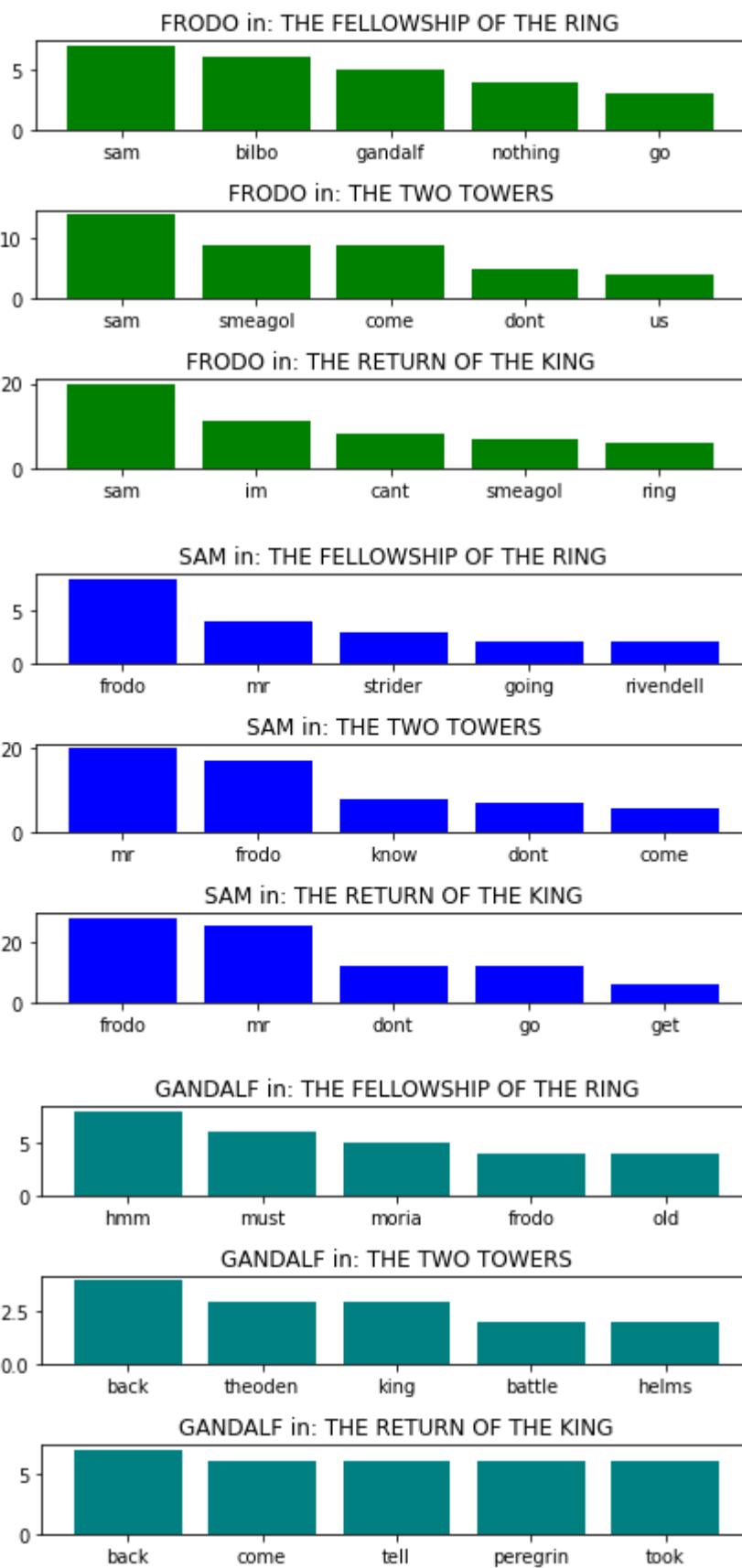
    # For third movie
    ax3.bar(range(len(subset_3)), values_3, tick_label=names_3, color=colors[i])
    ax3.set_title(f"{char} in: {df_list[2].movie.unique()[0]}")
    figure.tight_layout()

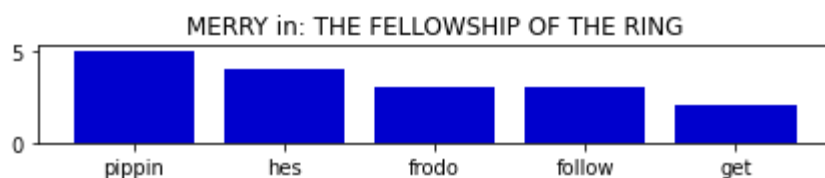
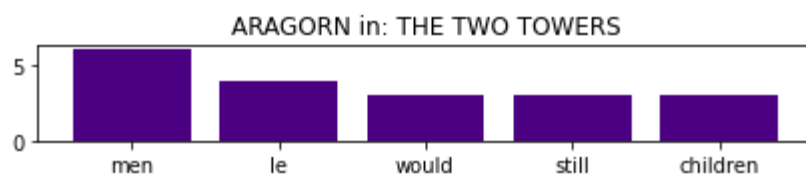
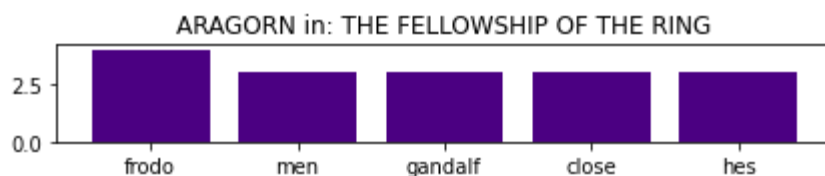
    plt.show()

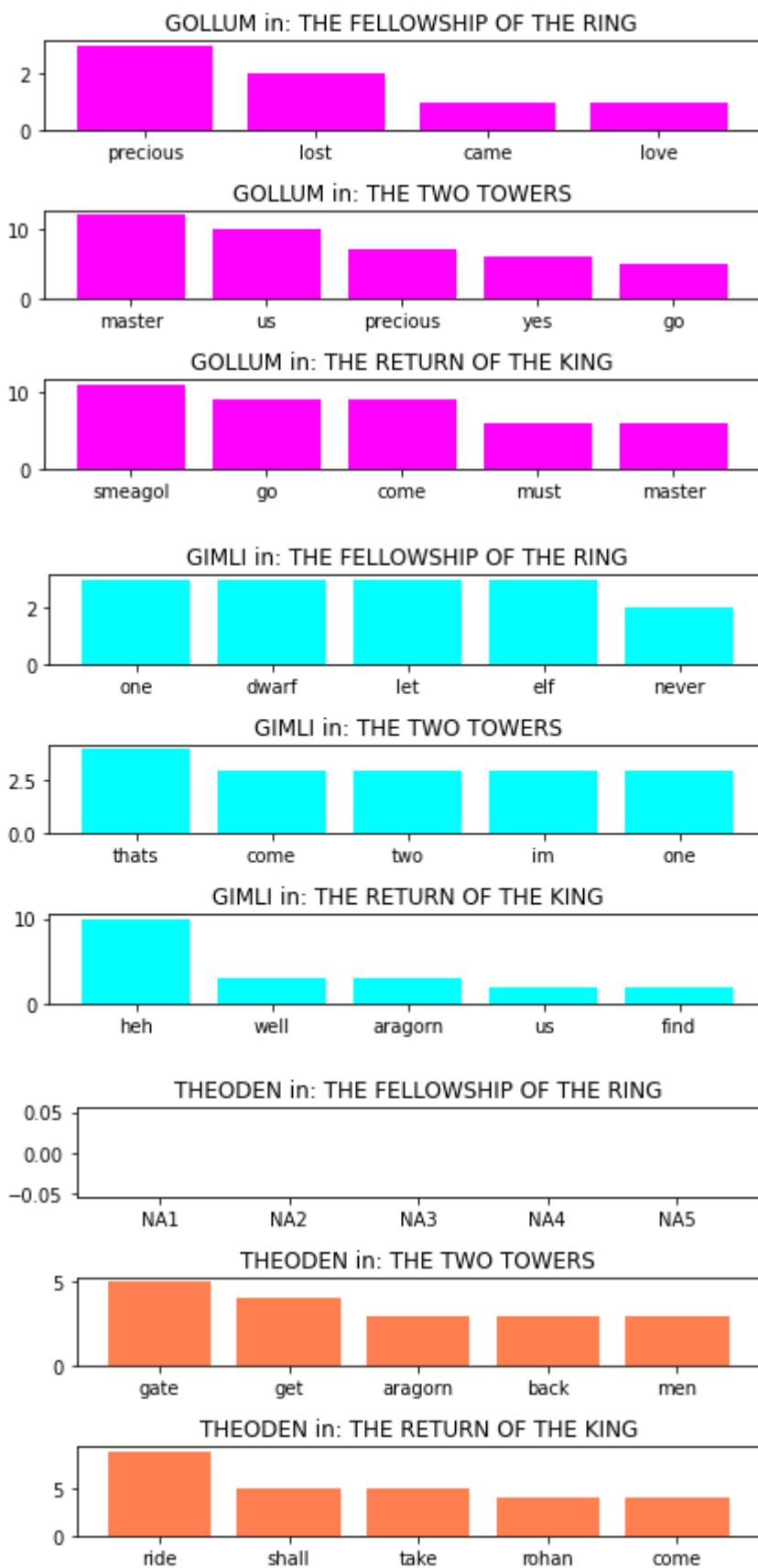
```

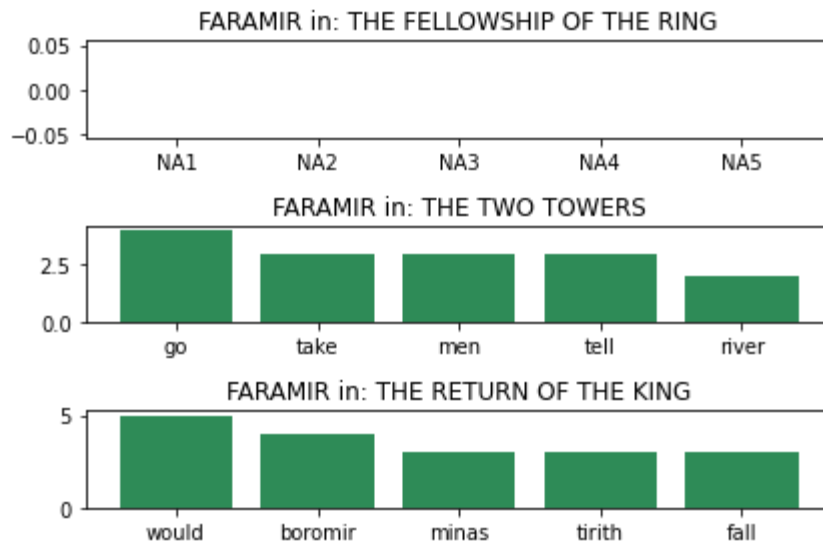


```
In [45]: for i in range(len(top10_chars_allmovies)):
          get_char_growth([fellowship_df,two_towers_df, return_king_df],
                           top10_chars_allmovies[i], i)
```









In []: