Import data from IMDB

In [1]:
```
import pandas as pd

pricipals_df = pd.read_csv("title.pricipals.tsv", sep="\t")
pricipals_df
```

Out[1]:

	tconst	ordering	nconst	category	job	characters
0	tt0000001	1	nm1588970	self	\N	["Self"]
1	tt0000001	2	nm0005690	director	\N	\N
2	tt0000001	3	nm0374658	cinematographer	director of photography	\N
3	tt0000002	1	nm0721526	director	\N	\N
4	tt0000002	2	nm1335271	composer	\N	\N
...	...	...	...	...	...	...
50014871	tt9916880	4	nm10535738	actress	\N	["Horrid Henry"]
50014872	tt9916880	5	nm0996406	director	principal director	\N
50014873	tt9916880	6	nm1482639	writer	\N	\N
50014874	tt9916880	7	nm2586970	writer	books	\N
50014875	tt9916880	8	nm1594058	producer	producer	\N

50014876 rows × 6 columns

In [2]:
```
basics_df = pd.read_csv("title.basics.tsv", sep="\t", low_memory=False)
basics_df
```

Out[2]:

	tconst	titleType	primaryTitle	originalTitle	isAdult	startYear	endYear	runtimeMi
0	tt0000001	short	Carmencita	Carmencita	0	1894	\N	
1	tt0000002	short	Le clown et ses chiens	Le clown et ses chiens	0	1892	\N	
2	tt0000003	short	Pauvre Pierrot	Pauvre Pierrot	0	1892	\N	
3	tt0000004	short	Un bon bock	Un bon bock	0	1892	\N	
4	tt0000005	short	Blacksmith Scene	Blacksmith Scene	0	1893	\N	
...	...	...	...	...	...	...	...	
8870617	tt9916848	tvEpisode	Episode #3.17	Episode #3.17	0	2010	\N	
8870618	tt9916850	tvEpisode	Episode #3.19	Episode #3.19	0	2010	\N	
8870619	tt9916852	tvEpisode	Episode #3.20	Episode #3.20	0	2010	\N	
8870620	tt9916856	short	The Wind	The Wind	0	2015	\N	
8870621	tt9916880	tvEpisode	Horrid Henry Knows It All	Horrid Henry Knows It All	0	2014	\N	

8870622 rows × 9 columns

In [3]:
```
name_df = pd.read_csv("name.basics.tsv", sep="\t")
name_df
```

Out[3]:

	nconst	primaryName	birthYear	deathYear	primaryProfession	
0	nm0000001	Fred Astaire	1899	1987	soundtrack,actor,miscellaneous	t
1	nm0000002	Lauren Bacall	1924	2014	actress,soundtrack	t
2	nm0000003	Brigitte Bardot	1934	\N	actress,soundtrack,music_department	t
3	nm0000004	John Belushi	1949	1982	actor,soundtrack,writer	t
4	nm0000005	Ingmar Bergman	1918	2007	writer,director,actor	t
...	...	...	...	...	...	
11582529	nm9993714	Romeo del Rosario	\N	\N	animation_department,art_department	
11582530	nm9993716	Essias Loberg	\N	\N	NaN	
11582531	nm9993717	Harikrishnan Rajan	\N	\N	cinematographer	
11582532	nm9993718	Aayush Nair	\N	\N	cinematographer	
11582533	nm9993719	Andre Hill	\N	\N	NaN	

11582534 rows × 6 columns

Data cleaning

In [4]:
```
# Only keep category column is "actor"
pricipals_df = pricipals_df[pricipals_df['category'] == 'actor']
# drop ordering, category, job and characters columns
pricipals_df = pricipals_df.drop(['ordering', 'category', 'job', 'characters'], axis
pricipals_df
```

Out[4]:

	tconst	nconst
11	tt0000005	nm0443482
12	tt0000005	nm0653042
16	tt0000007	nm0179163
17	tt0000007	nm0183947
21	tt0000008	nm0653028
...	...	...
50014851	tt9916852	nm5519557
50014852	tt9916852	nm8825009
50014862	tt9916856	nm10538646
50014868	tt9916880	nm1483166
50014870	tt9916880	nm0286175

11118202 rows × 2 columns

In [5]:
```
# drop titleType, originalTitle, isAdult, endYear, runtimeMinutes and genres columns
basics_df = basics_df.drop(['titleType', 'originalTitle', 'isAdult', 'endYear', 'run
# Filter data by startYear is "1990"
basics_df = basics_df[basics_df['startYear'] == '1990']
basics_df
```

Out[5]:

	tconst	primaryTitle	startYear
58205	tt0059325	Born in '45	1990
58759	tt0059900	Wenn du groß bist, lieber Adam	1990
63893	tt0065188	Vojtech, receny sirotek	1990
67130	tt0068494	Domo Arigato	1990
73695	tt0075259	Spy Story	1990
...	...	...	...
8868538	tt9912346	Jeanne Bourin	1990
8868552	tt9912376	Irina Ionesco	1990
8868554	tt9912380	Charles Hernu	1990
8870305	tt9916194	Episode dated 5 October 1990	1990
8870526	tt9916654	Episode dated 2 February 1990	1990

46819 rows × 3 columns

In [6]:
```
# drop birthYear, deathYear, primaryProfession, and knownForTitles columns
name_df = name_df.drop(['birthYear', 'deathYear', 'primaryProfession', 'knownForTitl
name_df
```

Out[6]:

	nconst	primaryName
0	nm0000001	Fred Astaire
1	nm0000002	Lauren Bacall
2	nm0000003	Brigitte Bardot
3	nm0000004	John Belushi
4	nm0000005	Ingmar Bergman
...	...	...
11582529	nm9993714	Romeo del Rosario
11582530	nm9993716	Essias Loberg
11582531	nm9993717	Harikrishnan Rajan
11582532	nm9993718	Aayush Nair
11582533	nm9993719	Andre Hill

11582534 rows × 2 columns

In [7]:
```
# Merge the three tables
df = pd.merge(left=basics_df,right=pricipals_df,on='tconst')
df = pd.merge(left=df,right=name_df,on='nconst')
# Export data to "hypergraph_data.csv"
df.to_csv('hypergraph_data.csv',index = False)
```

In [1]:
```
import pandas as pd

df = pd.read_csv('hypergraph_data.csv')
df
```

Out[1]:

	tconst	primaryTitle	startYear	nconst	primaryName
0	tt0059325	Born in '45	1990	nm0753957	Rolf Römer
1	tt0059325	Born in '45	1990	nm2535970	Paul Eichbaum
2	tt0059325	Born in '45	1990	nm0537007	Holger Mahlich
3	tt0099958	Die Kupferfalle	1990	nm0537007	Holger Mahlich
4	tt0059900	Wenn du groß bist, lieber Adam	1990	nm1052658	Stephan Jahnke
...	...	...	...	...	...
78431	tt9894646	The Lady from Rome Part 3	1990	nm7580066	Anthony M. Chin
78432	tt9894646	The Lady from Rome Part 3	1990	nm1199293	Giuliano Gensini
78433	tt9894646	The Lady from Rome Part 3	1990	nm0302380	Frank Gallagher
78434	tt9894880	Adiós a Cali	1990	nm2762404	Fernell Franco
78435	tt9894880	Adiós a Cali	1990	nm10529611	Óscar Muñoz

78436 rows × 5 columns

The COO representation

In [2]:
```
import numpy as np
import pandas as pd
import nwhy as nwhy
import copy
import os
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"

# Select all values of the tconst column from the dataframe
tconst = copy.copy(df.iloc[:,0].values)
# Select all values of the primaryTitle column from the dataframe
primaryTitle = copy.copy(df.iloc[:,1].values)
# Select all values of the nconst column from the dataframe
nconst = copy.copy(df.iloc[:,3].values)
# Select all values of the primaryName column from the dataframe
primaryName = copy.copy(df.iloc[:,4].values)

hyperedge_to_title_dic = dict()
title_to_hyperedge_dic = dict()
title_to_tconst_dic = dict()
tconst_dic = dict()
i = 0
j = 0
for item in tconst:
    if(not tconst_dic.__contains__(item)):
        tconst_dic[item] = i
        hyperedge_to_title_dic[i] = primaryTitle[j]
        if(not title_to_hyperedge_dic.__contains__(primaryTitle[j])):
            title_to_hyperedge_dic[primaryTitle[j]] = []
            title_to_tconst_dic[primaryTitle[j]] = []
        title_to_hyperedge_dic[primaryTitle[j]].append(i)
        title_to_tconst_dic[primaryTitle[j]].append(tconst[j])
        i += 1
    tconst[j] = tconst_dic[item]
    j += 1

vertex_to_name_dic = dict()
name_to_vertex_dic = dict()
name_to_nconst_dic = dict()
nconst_dic = dict()
i = 0
j = 0
for item in nconst:
    if(not nconst_dic.__contains__(item)):
        nconst_dic[item] = i
        vertex_to_name_dic[i] = primaryName[j]
        if(not name_to_vertex_dic.__contains__(primaryName[j])):
            name_to_vertex_dic[primaryName[j]] = []
            name_to_nconst_dic[primaryName[j]] = []
        name_to_vertex_dic[primaryName[j]].append(i)
        name_to_nconst_dic[primaryName[j]].append(nconst[j])
        i += 1
    nconst[j] = nconst_dic[item]
    j += 1

weight = [1] * tconst.size

# Row of sparse matrix of the hypergraph (hyperedges)
row = tconst
# Columns of sparse matrix of the hypergraph (vertices)
col = nconst
```

```
# Weights of sparse matrix of the hypergraph
data = np.array(weight)
```

Explore data relationships by manipulating the data table

Query 1: How many TV shows/movies in IMDB whose startYear is 1990?

In [3]:
```
def query_1_by_table():
    num_tv_movie_1990 = len(tconst_dic.keys())
    print('The number of TV shows/movies whose startYear is 1990: ', num_tv_movie_19
query_1_by_table()
```

```
The number of TV shows/movies whose startYear is 1990:  27612
```

Query 2: How many actors who have acted in TV shows/movies with startYear of 1990 in IMDB?

In [4]:
```
def query_2_by_table():
    num_actor_1990 = len(nconst_dic.keys())
    print('The number of actors who have acted in TV shows/movies with startYear of
query_2_by_table()
```

```
The number of actors who have acted in TV shows/movies with startYear of 1990:  2112
1
```

Query 3: Enter a TV show/movie with startYear of 1990 in IMDB, and query the number of actors in that TV show/movie.

In [5]:
```
title_tv_movie = input("Please enter a TV show/movie title: ")
def query_3_by_table(title_tv_movie):
    if(title_to_tconst_dic.__contains__(title_tv_movie)):
        # More than one line of output indicates that multiple movies have the same
        for tconst in title_to_tconst_dic[title_tv_movie]:
            temp_df = df[df['tconst'] == tconst]
            print('The number of actors in "', title_tv_movie, '": ', len(temp_df))
    else:
        print('Sorry, the TV show/movie "', title_tv_movie, '" was not found!')
query_3_by_table(title_tv_movie)
```

```
Please enter a TV show/movie title: Die Kupferfalle
The number of actors in " Die Kupferfalle ":  7
```

Query 4: Enter a TV show/movie with startYear of 1990 in IMDB, and query the actors in that TV show/movie.

In [6]:
```
title_tv_movie = input("Please enter a TV show/movie title: ")
def query_4_by_table(title_tv_movie):
    if(title_to_tconst_dic.__contains__(title_tv_movie)):
        # More than one block of output indicates that multiple movies have the same
        for tconst in title_to_tconst_dic[title_tv_movie]:
            temp_df = df[df['tconst'] == tconst]
            name_col = copy.copy(temp_df.iloc[:,4].values).tolist()
            for name in name_col:
                print('Actor', name_col.index(name) + 1, 'in "', title_tv_movie, '":
    else:
```

```
        print('Sorry, the TV show/movie "', title_tv_movie, '" was not found!')
query_4_by_table(title_tv_movie)
```

```
Please enter a TV show/movie title: Die Kupferfalle
Actor 1 in " Die Kupferfalle ": Holger Mahlich
Actor 2 in " Die Kupferfalle ": Ivan Desny
Actor 3 in " Die Kupferfalle ": Wolfgang Wahl
Actor 4 in " Die Kupferfalle ": Peter Bongartz
Actor 5 in " Die Kupferfalle ": Klaus J. Behrendt
Actor 6 in " Die Kupferfalle ": Vincenzo Benestante
Actor 7 in " Die Kupferfalle ": Ulrich Gebauer
```

Query 5: Enter an actor, and query the number of TV shows/movies with starYear of 1990 in which the actor is in.

In [7]:
```
name_actor = input("Please enter an actor: ")
def query_5_by_table(name_actor):
    if(name_to_nconst_dic.__contains__(name_actor)):
        # More than one line of output indicates that multiple actors have the same
        for nconst in name_to_nconst_dic[name_actor]:
            temp_df = df[df['nconst'] == nconst]
            print('The number of TV shows/movies with startYear of 1990 "', name_act
    else:
        print('Sorry, the actor "', name_actor, '" was not found!')
query_5_by_table(name_actor)
```

```
Please enter an actor: Holger Mahlich
The number of TV shows/movies with startYear of 1990 " Holger Mahlich " is in:  2
```

Query 6: Enter an actor, and query TV shows/movies with starYear of 1990 in which the actor is in.

In [8]:
```
name_actor = input("Please enter an actor: ")
def query_6_by_table(name_actor):
    if(name_to_nconst_dic.__contains__(name_actor)):
        # More than one block of output indicates that multiple actors have the same
        for nconst in name_to_nconst_dic[name_actor]:
            temp_df = df[df['nconst'] == nconst]
            title_col = copy.copy(temp_df.iloc[:,1].values).tolist()
            for title in title_col:
                print('TV show/movie', title_col.index(title) + 1, '"', name_actor,
    else:
        print('Sorry, the actor "', name_actor, '" was not found!')
query_6_by_table(name_actor)
```

```
Please enter an actor: Holger Mahlich
TV show/movie 1 " Holger Mahlich " is in:  Born in '45
TV show/movie 2 " Holger Mahlich " is in:  Die Kupferfalle
```

Create Hypergraph

In [9]:
```
# Create the hypergraph
h = nwhy.NWHypergraph(row, col, data)
print('Hypergraph created successfully!', h)
```

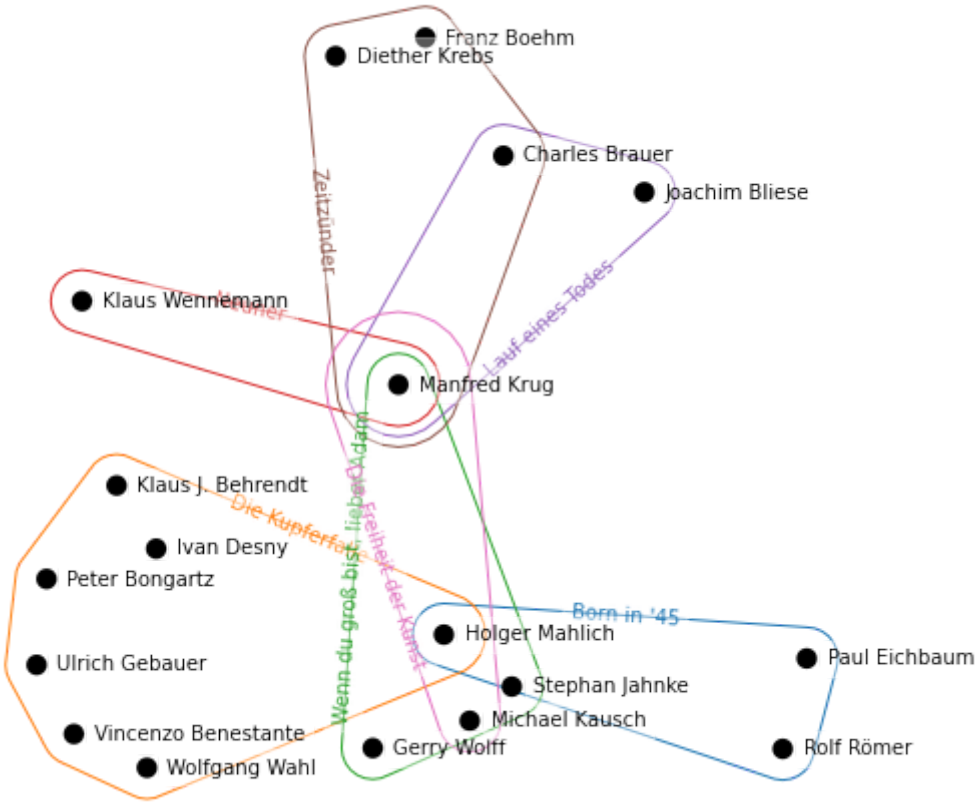
```
Hypergraph created successfully! <nwhy.NWHypergraph object at 0x7f1332a2aef0>
```

Visualize a part of the hypergraph through Hypernetx-Widget

In [10]:
```
import imp
import hypernetx as hnx

N_hyperedges = int(input("Please enter the number of hyperedges that you want to for
hyperedges = np.arange(N_hyperedges)
scenes = dict()
for hyperedge in hyperedges:
    title = hyperedge_to_title_dic[hyperedge]
    scenes[title] = []
    vertices = h.edge_incidence(hyperedge)
    for vertex in vertices:
        name = vertex_to_name_dic[vertex]
        scenes[title].append(name)
    scenes[title] = tuple(scenes[title])

H = hnx.Hypergraph(scenes)
hnx.draw(H)
```

```
/tmp/ipykernel_229/3587506147.py:1: DeprecationWarning: the imp module is deprecated
in favour of importlib; see the module's documentation for alternative uses
  import imp
/usr/local/lib/python3.9/dist-packages/numpy/core/getlimits.py:499: UserWarning: The
value of the smallest subnormal for <class 'numpy.float64'> type is zero.
  setattr(self, word, getattr(machar, word).flat[0])
/usr/local/lib/python3.9/dist-packages/numpy/core/getlimits.py:89: UserWarning: The
value of the smallest subnormal for <class 'numpy.float64'> type is zero.
  return self._float_to_str(self.smallest_subnormal)
/usr/local/lib/python3.9/dist-packages/numpy/core/getlimits.py:499: UserWarning: The
value of the smallest subnormal for <class 'numpy.float32'> type is zero.
  setattr(self, word, getattr(machar, word).flat[0])
/usr/local/lib/python3.9/dist-packages/numpy/core/getlimits.py:89: UserWarning: The
value of the smallest subnormal for <class 'numpy.float32'> type is zero.
  return self._float_to_str(self.smallest_subnormal)
```

```
Please enter the number of hyperedges that you want to form a part of the hypergrap
h: 7
```



NWHypergraph class methods:

In [12]:
```
# NWHypergraph class methods:

# print('-- collapsing edges without returning equal class')
# equal_class = h.collapse_edges()
# print(equal_class)

# print('-- collapsing nodes without returning equal class')
# equal_class = h.collapse_nodes()
# print(equal_class)

# print('-- collapsing nodes and edges without returning equal class')
# equal_class = h.collapse_nodes_and_edges()
# print(equal_class)

# print('-- collapsing edges with returning equal class')
# equal_class = h.collapse_edges(return_equivalence_class=True)
# print(equal_class)

# print('-- collapsing nodes with returning equal class')
# equal_class = h.collapse_nodes(return_equivalence_class=True)
# print(equal_class)

# print('-- collapsing nodes and edges with returning equal class')
# equal_class = h.collapse_nodes_and_edges(return_equivalence_class=True)
# print(equal_class)

# print('-- edge_size_dist()')
# equal_class = h.edge_size_dist()
# print(equal_class)

# print('-- node_size_dist()')
# equal_class = h.node_size_dist()
# print(equal_class)

# print('-- edge_incidence(edge)')
# equal_class = h.edge_incidence(666)
# print(equal_class)

# print('-- node_incidence(node)')
# equal_class = h.node_incidence(666)
# print(equal_class)

# print('-- degree(node, min_size=1, max_size=None)')
# equal_class = h.degree(666, min_size=1, max_size=None)
# print(equal_class)

# print('-- size(edge, min_degree=1, max_degree=None)')
# equal_class = h.size(666, min_degree=1, max_degree=None)
# print(equal_class)

# print('-- dim(edge)')
# equal_class = h.dim(666)
# print(equal_class)

# print('-- number_of_nodes()')
# equal_class = h.number_of_nodes()
# print(equal_class)

# print('-- number_of_edges()')
# equal_class = h.number_of_edges()
# print(equal_class)
```

```
# print('-- singletons()')
# equal_class = h.singletons()
# print(equal_class)

# print('-- toplexes()')
# equal_class = h.toplexes()
# print(equal_class)

# print('-- s_linegraph(s=1, edges=True)')
# equal_class = h.s_linegraph(s=1, edges=True)
# print(equal_class)

# print('-- s_linegraphs(1, edges=True)')
# equal_class = h.s_linegraphs([1,2,3,4,5,6], edges=True)
# print(equal_class)
```

Hypergraph application(analysis)

Query 1: How many TV shows/movies in IMDB whose startYear is 1990?

In [11]:
```
def query_1_by_hypergraph():
    num_tv_movie_1990 = h.number_of_edges()
    print('The number of TV shows/movies whose startYear is 1990: ', num_tv_movie_19
query_1_by_hypergraph()
```

```
The number of TV shows/movies whose startYear is 1990:  27612
```

Query 2: How many actors who have acted in TV shows/movies with startYear of 1990 in IMDB?

In [12]:
```
def query_2_by_hypergraph():
    num_actor_1990 = h.number_of_nodes()
    print('The number of actors who have acted in TV shows/movies with startYear of
query_2_by_hypergraph()
```

```
The number of actors who have acted in TV shows/movies with startYear of 1990:  2112
1
```

Query 3: Enter a TV show/movie with startYear of 1990 in IMDB, and query the number of actors in that TV show/movie.

In [13]:
```
title_tv_movie = input("Please enter a TV show/movie title: ")
def query_3_by_hypergraph(title_tv_movie):
    if(title_to_hyperedge_dic.__contains__(title_tv_movie)):
        # More than one line of output indicates that multiple movies have the same
        for hyperedge in title_to_hyperedge_dic[title_tv_movie]:
            num_actor_tv_movie = h.size(hyperedge, min_degree=1, max_degree=None)
            print('The number of actors in "', title_tv_movie, '": ', num_actor_tv_m
    else:
        print('Sorry, the TV show/movie "', title_tv_movie, '" was not found!')
query_3_by_hypergraph(title_tv_movie)
```

```
Please enter a TV show/movie title: Die Kupferfalle
The number of actors in " Die Kupferfalle ":  7
```

Query 4: Enter a TV show/movie with startYear of 1990 in

IMDB, and query the actors in that TV show/movie.

In [14]:
```
title_tv_movie = input("Please enter a TV show/movie title: ")
def query_4_by_hypergraph(title_tv_movie):
    if(title_to_hyperedge_dic.__contains__(title_tv_movie)):
        # More than one block of output indicates that multiple movies have the same
        for hyperedge in title_to_hyperedge_dic[title_tv_movie]:
            vertices = h.edge_incidence(hyperedge)
            for vertex in vertices:
                name = vertex_to_name_dic[vertex]
                print('Actor', vertices.index(vertex) + 1, 'in "', title_tv_movie, '
    else:
        print('Sorry, the TV show/movie "', title_tv_movie, '" was not found!')
query_4_by_hypergraph(title_tv_movie)
```

```
Please enter a TV show/movie title: Die Kupferfalle
Actor 1 in " Die Kupferfalle ": Holger Mahlich
Actor 2 in " Die Kupferfalle ": Ivan Desny
Actor 3 in " Die Kupferfalle ": Wolfgang Wahl
Actor 4 in " Die Kupferfalle ": Peter Bongartz
Actor 5 in " Die Kupferfalle ": Klaus J. Behrendt
Actor 6 in " Die Kupferfalle ": Vincenzo Benestante
Actor 7 in " Die Kupferfalle ": Ulrich Gebauer
```

Query 5: Enter an actor, and query the number of TV shows/movies with starYear of 1990 in which the actor is in.

In [15]:
```
name_actor = input("Please enter an actor: ")
def query_5_by_hypergraph(name_actor):
    if(name_to_vertex_dic.__contains__(name_actor)):
        # More than one line of output indicates that multiple actors have the same
        for vertex in name_to_vertex_dic[name_actor]:
            num_tv_movie_actor = h.degree(vertex, min_size=1, max_size=None)
            print('The number of TV shows/movies with startYear of 1990 "', name_act
    else:
        print('Sorry, the actor "', name_actor, '" was not found!')
query_5_by_hypergraph(name_actor)
```

```
Please enter an actor: Holger Mahlich
The number of TV shows/movies with startYear of 1990 " Holger Mahlich " is in:  2
```

Query 6: Enter an actor, and query TV shows/movies with starYear of 1990 in which the actor is in.

In [16]:
```
name_actor = input("Please enter an actor: ")
def query_6_by_hypergraph(name_actor):
    if(name_to_vertex_dic.__contains__(name_actor)):
        # More than one block of output indicates that multiple actors have the same
        for vertex in name_to_vertex_dic[name_actor]:
            hyperedges = h.node_incidence(vertex)
            for hyperedge in hyperedges:
                title = hyperedge_to_title_dic[hyperedge]
                print('TV show/movie', hyperedges.index(hyperedge) + 1, '"', name_ac
    else:
        print('Sorry, the actor "', name_actor, '" was not found!')
query_6_by_hypergraph(name_actor)
```

```
Please enter an actor: Holger Mahlich
TV show/movie 1 " Holger Mahlich " is in:  Born in '45
TV show/movie 2 " Holger Mahlich " is in:  Die Kupferfalle
```

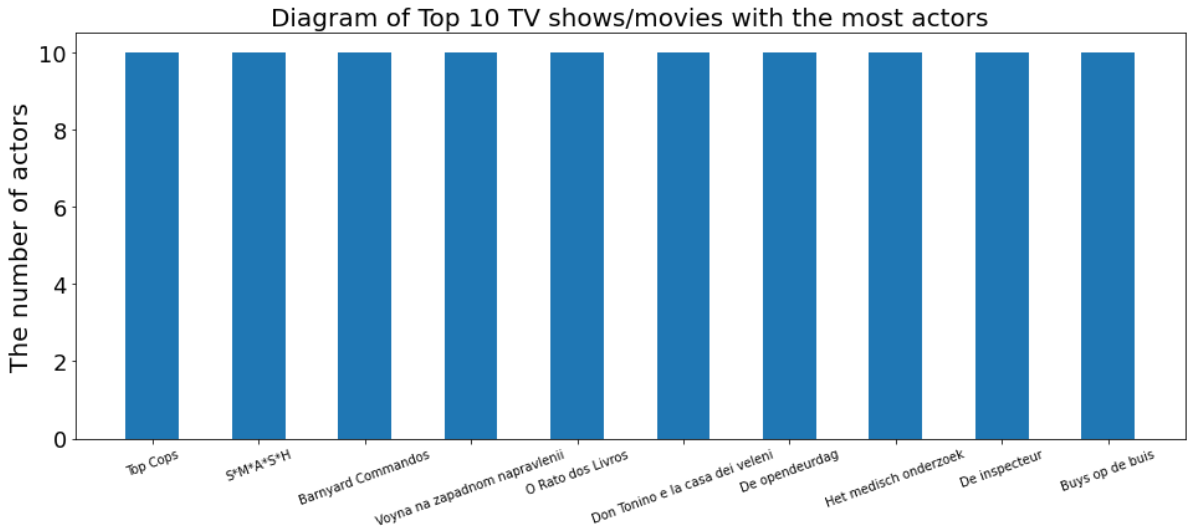
Query 7: Find the top N TV shows/movies with the most actors

In [17]:
```
N = int(input("Please enter the value of N: "))
num_vertices_arr = h.edge_size_dist()
desc_num_vertices_arr = sorted(num_vertices_arr, reverse = True)
N_desc_num_vertices_arr = desc_num_vertices_arr[:N]
print('Top #\tTV show/movie title\tthe number of actors\tactors\n')
i = 1
title_arr = []
num_actor_arr = []
for item in N_desc_num_vertices_arr:
    idx = num_vertices_arr.index(item)
    num_vertices_arr[idx] = -1
    title = hyperedge_to_title_dic[idx]
    title_arr.append(title)
    num_actor_arr.append(item)
    print('Top', i, '\t', title, '\t\t', item, end='\t')
    vertices = h.edge_incidence(idx)
    for vertex in vertices:
        name = vertex_to_name_dic[vertex]
        print(name,end = ', ')
    print('\n')
    i += 1
```

```
Please enter the value of N: 10
Top #   TV show/movie title     the number of actors    actors

Top 1    Top Cops                10     Steve Adams, Jeffrey Knight, Chris Potter, V
ictor Ertmanis, Matt Cooke, Bill Lake, Joseph Griffin, William Colgate, Allan Clow,
Timm Zemanek,

Top 2    S*M*A*S*H               10     Tor Isedal, Måns Herngren, Svante Grundberg,
Peter Wahlbeck, Felix Herngren, Sten Ljunggren, Tomas Norström, Claes Månsson, Per E
ggers, Björn Granath,

Top 3    Barnyard Commandos              10     Pat Fraley, Thom Bray, S. Scott Bull
ock, Marc François, Paul Kreppel, John Mariano, Robert Ridgely, Lennie Weinrib, Dann
y Wells, Jacques Ferrière,

Top 4    Voyna na zapadnom napravlenii           10     Oleg Savkin, Nikolay Oleyni
k, Viktor Stepanov, Archil Gomiashvili, Mikhail Ulyanov, Igor Taradaikin, Pavel Moro
zenko, Pavel Makhotin, Vitali Rosstalnoy, Nikolay Zasukhin,

Top 5    O Rato dos Livros               10     José Eduardo, Rui Anjos, Orlando Cos
ta, Fernando Gomes, António Feio, Quim Cachopo, Carlos Alberto Moniz, Waldemar de So
uza, João Azevedo, Almeno Gonçalves,

Top 6    Don Tonino e la casa dei veleni                 10     Giampiero Bianchi, A
ndrea Roncato, Gigi Sammarchi, Cesare Gallarini, Marco Bechini, Tommy Bianco, Sante
Calogero, Salvatore Ciraolo, Marco Della Noce, Danilo Fernandez,

Top 7    De opendeurdag                  10     Jacques Vermeire, Chris Cauwenbergh
s, Frank Dingenen, Guido Lauwaert, Vic Moeremans, Günther Lesage, Hans Cardijn, Bart
De Pauw, Wouter Denayer, Wim Geysen,

Top 8    Het medisch onderzoek           10     Jacques Vermeire, Frank Dingenen, Gu
ido Lauwaert, Günther Lesage, Hans Cardijn, Bart De Pauw, Wouter Denayer, Wim Geyse
n, Rob Renckens, Gert Nevens,

Top 9    De inspecteur           10     Jacques Vermeire, Koen Crucke, Chris Cauwenb
erghs, Frank Dingenen, Rudi Delhem, Hans Cardijn, Bart De Pauw, Wouter Denayer, Stan
y Crets, Koen De Cauter,

Top 10   Buys op de buis                 10     Jacques Vermeire, Koen Crucke, Chris
Cauwenberghs, Frank Dingenen, Günther Lesage, Hans Cardijn, Bart De Pauw, Wouter Den
ayer, Wim Geysen, Emiel Goelen,
```

In [18]:

```
import matplotlib.pyplot as plt
x = title_arr
y = num_actor_arr
plt.figure(figsize=(16, 6))
plt.bar(x, y, width=0.5)
plt.xticks(fontsize=10, rotation=20)
plt.yticks(fontsize=18)
plt.ylabel(u'The number of actors', fontsize=20)
diagram_title = str(N).join(['Diagram of Top ',' TV shows/movies with the most actor
plt.title(diagram_title, fontsize=20)
plt.show()
```



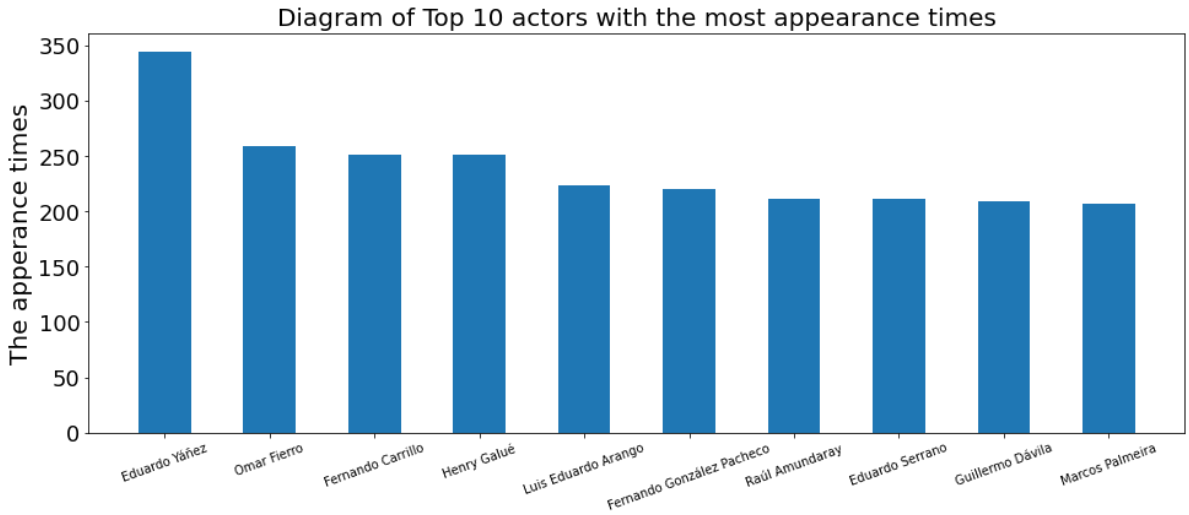
Query 8: Find the top N actors with the most appearance times

In [19]:
```
N = int(input("Please enter the value of N: "))
num_hyperedges_arr = h.node_size_dist()
desc_num_hyperedges_arr = sorted(num_hyperedges_arr, reverse = True)
N_desc_num_hyperedges_arr = desc_num_hyperedges_arr[:N]
print('Top #\tname\tthe number of TV shows/movies\tTV shows/movies\n')
i = 1
name_arr = []
num_tv_movie_arr = []
for item in N_desc_num_hyperedges_arr:
    idx = num_hyperedges_arr.index(item)
    num_hyperedges_arr[idx] = -1
    name = vertex_to_name_dic[idx]
    name_arr.append(name)
    num_tv_movie_arr.append(item)
    print('Top', i, '\t', name, '\t\t', item, end='\t')
    hyperedges = h.node_incidence(idx)
    for hyperedge in hyperedges:
        if(hyperedges.index(hyperedge) == 10):
            print('......')
            break
        title = hyperedge_to_title_dic[hyperedge]
        print(title,end = ', ')
    print('\n')
    i += 1
```

```
Please enter the value of N: 10
Top #   name    the number of TV shows/movies   TV shows/movies

Top 1    Eduardo Yáñez           344    Yo compro esa mujer, Episode #1.4, Episode #
1.6, Episode #1.7, Episode #1.8, Episode #1.9, Episode #1.10, Episode #1.5, Episode
#1.14, Episode #1.17, ......


Top 2    Omar Fierro             259    Lo inesperado, Cuando llega el amor, Furia a
sesina, Mi pequeña Soledad, Episode #1.4, Episode #1.5, Episode #1.6, Episode #1.7,
Episode #1.8, Episode #1.9, ......


Top 3    Fernando Carrillo               251    Pasionaria, Episode #1.4, Episode #
1.5, Episode #1.6, Episode #1.7, Episode #1.8, Episode #1.9, Episode #1.10, Episode
#1.11, Episode #1.12, ......


Top 4    Henry Galué             251    Pasionaria, Episode #1.4, Episode #1.5, Epis
ode #1.6, Episode #1.7, Episode #1.8, Episode #1.9, Episode #1.10, Episode #1.11, Ep
isode #1.12, ......


Top 5    Luis Eduardo Arango             223    Música, maestro, Episode #2.60, Epis
ode #1.4, Episode #1.6, Episode #1.5, Episode #1.7, Episode #1.9, Episode #1.8, Epis
ode #1.11, Episode #1.10, ......


Top 6    Fernando González Pacheco               220    Música, maestro, Episode #2.
60, Episode #1.4, Episode #1.6, Episode #1.5, Episode #1.7, Episode #1.9, Episode #
1.8, Episode #1.11, Episode #1.10, ......


Top 7    Raúl Amundaray                  211    Emperatriz, Episode #1.2, Episode #
1.3, Episode #1.4, Episode #1.6, Episode #1.5, Episode #1.7, Episode #1.8, Episode #
1.9, Episode #1.10, ......


Top 8    Eduardo Serrano                 211    Emperatriz, Episode #1.2, Episode #
1.3, Episode #1.4, Episode #1.6, Episode #1.5, Episode #1.7, Episode #1.8, Episode #
1.9, Episode #1.10, ......


Top 9    Guillermo Dávila                209    Adorable Monica, Episode #1.4, Episo
de #1.5, Episode #1.6, Episode #1.7, Episode #1.8, Episode #1.9, Episode #1.10, Epis
ode #1.11, Episode #1.13, ......


Top 10   Marcos Palmeira                207    Carnaval, Desire, Pantanal, Stelinh
a, Barrela: Escola de Crimes, Making of 'Pantanal', Episode #1.7, Episode #1.8, Epis
ode #1.9, Episode #1.10, ......
```

In [20]:

```
x = name_arr
y = num_tv_movie_arr
plt.figure(figsize=(16, 6))
plt.bar(x, y, width=0.5)
plt.xticks(fontsize=10, rotation=20)
plt.yticks(fontsize=18)
plt.ylabel(u'The apperance times', fontsize=20)
diagram_title = str(N).join(['Diagram of Top ',' actors with the most appearance tim
plt.title(diagram_title, fontsize=20)
plt.show()
```



Query 9: Find movies with only one actor in it, and that actor only acted in that movie.

In [21]:
```
singleton_hyperedges = h.singletons()
if(len(singleton_hyperedges) > 0):
    print('title\t\t\t\t\t\t\tactor\n')
else:
    print('There is no such movie.')
for hyperedge in singleton_hyperedges:
    if(singleton_hyperedges.index(hyperedge) == 30):
        print('......')
        break
    title = hyperedge_to_title_dic[hyperedge]
    print(title, end = '\t\t\t\t\t\t\t')
    vertices = h.edge_incidence(hyperedge)
    name = vertex_to_name_dic[vertices[0]]
    print(name)
```

```
title                                                    actor

Dragon Hunt                                                    B. Bob
Man Eaters                                                     Daniel Colas
Sometime in August                                                     Craig Lorime
r
The American Angels: Baptism of Blood
Tray Loren
Cézanne - Conversation with Joachim Gasquet
Jean-Marie Straub
Les noces de papier                                                    Manuel Arang
uiz
Ono                                                      Rolan Bykov
Party Girls                                                    Kurt Woodruff
A Woman's Revenge                                                      Jean-Louis M
urat
Forbidden Love                                                 Hans-Peter Dahm
America Tonight                                                Fred Melamed
The Crystal Maze                                                       Adam Buxton
Kevin Seal, Sporting Fool                                                      Kevi
n Seal
Marleneken                                                     Wolfried Lier
On Scene: Emergency Response                                           Dave
Forman
Overtime... with Pat O'Brien                                           Pat
O'Brien
Alan Alda in Scientific American Frontiers
John Osborne
Video Challenge                                                Steve Kelley
All the Vermeers in New York                                           Step
hen Lack
American Dream                                                 Juan Munoz
Artikos                                                  Nelson Villagra
Aschenglut                                                     Mathieu Carrière
Baito                                                    Jan Dop
Basket Case 2                                                  Kevin Van Hentenryck
Die Beichte                                                    Jochen Kuhn
Best Shots                                                     Mick Best
Der Blasi                                                      Reinhard Nowak
Blue Bayou                                                     Michael Audley
Blue Planet                                                    James Buchli
Le blé en herbe                                                Matthieu Rozé
......
```

Comparisons of results and runtime between NWhy lib and the approach to manipulate the data table

Comparison 1 (Query 1)

In [22]:
```
from time import time

print('Query 1:')
print('\n--- Query 1 through manipulating the data table ---')
t_start=time()
query_1_by_table()
t_end=time()
t_cost=t_end-t_start
print('Manipulating the data table takes %0.8f seconds'%t_cost)

print('\n--- Query 1 through NWhy lib ---')
t_start=time()
```

```
query_1_by_hypergraph()
t_end=time()
t_cost=t_end-t_start
print('NWhy lib takes %0.8f seconds'%t_cost)

del t_end,t_start,t_cost
```

```
Query 1:

--- Query 1 through manipulating the data table ---
The number of TV shows/movies whose startYear is 1990:  27612
Manipulating the data table takes 0.00009036 seconds

--- Query 1 through NWhy lib ---
The number of TV shows/movies whose startYear is 1990:  27612
NWhy lib takes 0.00006247 seconds
```

Comparison 2 (Query 2)

In [23]:
```
print('Query 2:')
print('\n--- Query 2 through manipulating the data table ---')
t_start=time()
query_2_by_table()
t_end=time()
t_cost=t_end-t_start
print('Manipulating the data table takes %0.8f seconds'%t_cost)

print('\n--- Query 2 through NWhy lib ---')
t_start=time()
query_2_by_hypergraph()
t_end=time()
t_cost=t_end-t_start
print('NWhy lib takes %0.8f seconds'%t_cost)

del t_end,t_start,t_cost
```

```
Query 2:

--- Query 2 through manipulating the data table ---
The number of actors who have acted in TV shows/movies with startYear of 1990:  2112
1
Manipulating the data table takes 0.00005674 seconds

--- Query 2 through NWhy lib ---
The number of actors who have acted in TV shows/movies with startYear of 1990:  2112
1
NWhy lib takes 0.00005627 seconds
```

Comparison 3 (Query 3)

In [24]:
```
print('Query 3:\n')
title_tv_movie = input("Please enter a TV show/movie title: ")
print('\n--- Query 3 through manipulating the data table ---')
t_start=time()
query_3_by_table(title_tv_movie)
t_end=time()
t_cost=t_end-t_start
print('Manipulating the data table takes %0.8f seconds'%t_cost)

print('\n--- Query 3 through NWhy lib ---')
t_start=time()
query_3_by_hypergraph(title_tv_movie)
```

```
t_end=time()
t_cost=t_end-t_start
print('NWhy lib takes %0.8f seconds'%t_cost)

del t_end,t_start,t_cost
```

```
Query 3:

Please enter a TV show/movie title: Die Kupferfalle

--- Query 3 through manipulating the data table ---
The number of actors in " Die Kupferfalle ":  7
Manipulating the data table takes 0.00883031 seconds

--- Query 3 through NWhy lib ---
The number of actors in " Die Kupferfalle ":  7
NWhy lib takes 0.00013185 seconds
```

Comparison 4 (Query 4)

In [25]:
```
print('Query 4:\n')
title_tv_movie = input("Please enter a TV show/movie title: ")
print('\n--- Query 4 through manipulating the data table ---')
t_start=time()
query_4_by_table(title_tv_movie)
t_end=time()
t_cost=t_end-t_start
print('Manipulating the data table takes %0.8f seconds'%t_cost)

print('\n--- Query 4 through NWhy lib ---')
t_start=time()
query_4_by_hypergraph(title_tv_movie)
t_end=time()
t_cost=t_end-t_start
print('NWhy lib takes %0.8f seconds'%t_cost)

del t_end,t_start,t_cost
```

```
Query 4:

Please enter a TV show/movie title: Die Kupferfalle

--- Query 4 through manipulating the data table ---
Actor 1 in " Die Kupferfalle ": Holger Mahlich
Actor 2 in " Die Kupferfalle ": Ivan Desny
Actor 3 in " Die Kupferfalle ": Wolfgang Wahl
Actor 4 in " Die Kupferfalle ": Peter Bongartz
Actor 5 in " Die Kupferfalle ": Klaus J. Behrendt
Actor 6 in " Die Kupferfalle ": Vincenzo Benestante
Actor 7 in " Die Kupferfalle ": Ulrich Gebauer
Manipulating the data table takes 0.01185775 seconds

--- Query 4 through NWhy lib ---
Actor 1 in " Die Kupferfalle ": Holger Mahlich
Actor 2 in " Die Kupferfalle ": Ivan Desny
Actor 3 in " Die Kupferfalle ": Wolfgang Wahl
Actor 4 in " Die Kupferfalle ": Peter Bongartz
Actor 5 in " Die Kupferfalle ": Klaus J. Behrendt
Actor 6 in " Die Kupferfalle ": Vincenzo Benestante
Actor 7 in " Die Kupferfalle ": Ulrich Gebauer
NWhy lib takes 0.00027299 seconds
```

Comparison 5 (Query 5)

In [26]:
```
print('Query 5:\n')
name_actor = input("Please enter an actor: ")
print('\n--- Query 5 through manipulating the data table ---')
t_start=time()
query_5_by_table(name_actor)
t_end=time()
t_cost=t_end-t_start
print('Manipulating the data table takes %0.8f seconds'%t_cost)

print('\n--- Query 5 through NWhy lib ---')
t_start=time()
query_5_by_hypergraph(name_actor)
t_end=time()
t_cost=t_end-t_start
print('NWhy lib takes %0.8f seconds'%t_cost)

del t_end,t_start,t_cost
```

```
Query 5:

Please enter an actor: Holger Mahlich

--- Query 5 through manipulating the data table ---
The number of TV shows/movies with startYear of 1990 " Holger Mahlich " is in:  2
Manipulating the data table takes 0.00828266 seconds

--- Query 5 through NWhy lib ---
The number of TV shows/movies with startYear of 1990 " Holger Mahlich " is in:  2
NWhy lib takes 0.00028610 seconds
```

Comparison 6 (Query 6)

In [27]:
```
print('Query 6:\n')
name_actor = input("Please enter an actor: ")
print('\n--- Query 6 through manipulating the data table ---')
t_start=time()
query_6_by_table(name_actor)
t_end=time()
t_cost=t_end-t_start
print('Manipulating the data table takes %0.8f seconds'%t_cost)

print('\n--- Query 6 through NWhy lib ---')
t_start=time()
query_6_by_hypergraph(name_actor)
t_end=time()
t_cost=t_end-t_start
print('NWhy lib takes %0.8f seconds'%t_cost)

del t_end,t_start,t_cost
```

```
Query 6:

Please enter an actor: Holger Mahlich

--- Query 6 through manipulating the data table ---
TV show/movie 1 " Holger Mahlich " is in:  Born in '45
TV show/movie 2 " Holger Mahlich " is in:  Die Kupferfalle
Manipulating the data table takes 0.00981736 seconds

--- Query 6 through NWhy lib ---
TV show/movie 1 " Holger Mahlich " is in:  Born in '45
TV show/movie 2 " Holger Mahlich " is in:  Die Kupferfalle
NWhy lib takes 0.00011110 seconds
```

Create Slinegraph

In [28]:
```
# Create the slinegraph
s = h.s_linegraph(s=1, edges=True)
print('Slinegraph created successfully! (s=1)', s)
```

```
# [ to_two_graph_count_neighbors_cyclic ]: 297 ms
Slinegraph created successfully! (s=1) <nwhy.Slinegraph object at 0x7f12f7067e30>
# [ populate_linegraph_from_neighbor_map ]: 45 ms
linegraph size: 891400
```

Slinegraph class methods:

In [25]:
```
# Slinegraph class methods:

# print('-- get_singletons()')
# equal_class = s.get_singletons()
# print(equal_class)

# print('-- s_connected_components()')
# equal_class = s.s_connected_components()
# print(equal_class)

# print('-- is_s_connected()')
# equal_class = s.is_s_connected()
# print(equal_class)

# print('-- s_distance(src, dest)')
# equal_class = s.s_distance(src, dest)
# print(equal_class)

# print('-- s_diameter(src, dest)')
# equal_class = s.s_diameter(src, dest)
# print(equal_class)

# print('-- s_path(src, dest)')
# equal_class = s.s_path(src, dest)
# print(equal_class)

# print('-- s_betweenness_centrality(normalized=True)')
# equal_class = s.s_betweenness_centrality(normalized=True)
# print(equal_class)

# print('-- s_closeness_centrality(v=None)')
# equal_class = s.s_closeness_centrality(v=None)
# print(equal_class)
```

```
# print('-- s_harmonic_closeness_centrality(v=None)')
# equal_class = s.s_harmonic_closeness_centrality(v=None)
# print(equal_class)

# print('-- s_eccentricity(v=None)')
# equal_class = s.s_eccentricity(v=None)
# print(equal_class)

# print('-- s_neighbors(v)')
# equal_class = s.s_neighbors(v)
# print(equal_class)

# print('-- s_degree(v)')
# equal_class = s.s_degree(v)
# print(equal_class)
```

S-line graphs construction

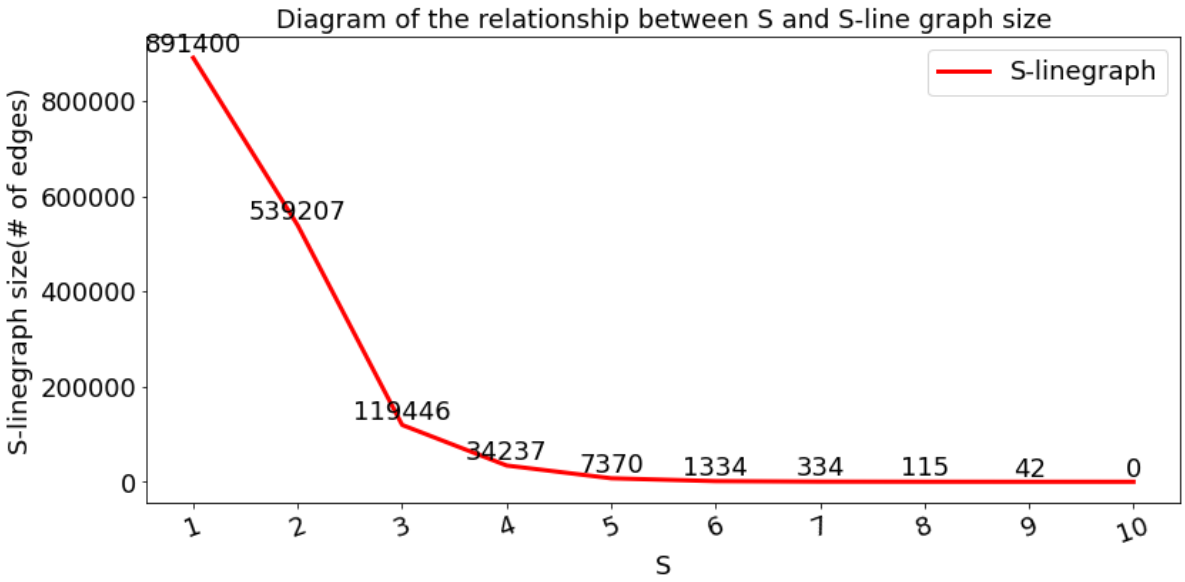
In [29]:
```
s_line_graph_arr = h.s_linegraphs([1,2,3,4,5,6,7,8,9,10], edges=True)
print(s_line_graph_arr)
```

```
# [ populate_linegraph_from_neighbor_map ]: 17 ms
[<nwhy.Slinegraph object at 0x7f12f57b07f0>, <nwhy.Slinegraph object at 0x7f12f57b08
70>, <nwhy.Slinegraph object at 0x7f12f57b04b0>, <nwhy.Slinegraph object at 0x7f12f5
71d630>, <nwhy.Slinegraph object at 0x7f12f571d8b0>, <nwhy.Slinegraph object at 0x7f
12f57acc70>, <nwhy.Slinegraph object at 0x7f12f57accb0>, <nwhy.Slinegraph object at
0x7f12f57ac5f0>, <nwhy.Slinegraph object at 0x7f12f710db70>, <nwhy.Slinegraph object
at 0x7f12f57ac130>]
linegraph size: 891400
# [ populate_linegraph_from_neighbor_map ]: 21 ms
linegraph size: 539207
# [ populate_linegraph_from_neighbor_map ]: 14 ms
linegraph size: 119446
# [ populate_linegraph_from_neighbor_map ]: 13 ms
linegraph size: 34237
# [ populate_linegraph_from_neighbor_map ]: 13 ms
linegraph size: 7370
# [ populate_linegraph_from_neighbor_map ]: 12 ms
linegraph size: 1334
# [ populate_linegraph_from_neighbor_map ]: 12 ms
linegraph size: 334
# [ populate_linegraph_from_neighbor_map ]: 12 ms
linegraph size: 115
# [ populate_linegraph_from_neighbor_map ]: 13 ms
linegraph size: 42
# [ populate_linegraph_from_neighbor_map ]: 12 ms
linegraph size: 0
```

Diagram of the relationship between S and S-linegraph size

In [30]:
```
x_ticks = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
x = np.arange(len(x_ticks))
y = [891400, 539207, 119446, 34237, 7370, 1334, 334, 115, 42, 0]

plt.figure(figsize=(13, 6))
plt.plot(x, y, color='#FF0000', label='S-linegraph', linewidth=3.0)
for a, b in zip(x, y):
    plt.text(a, b, '%d'%b, ha='center', va= 'bottom', fontsize=18)
```

```
plt.xticks([r for r in x], x_ticks, fontsize=18, rotation=20)
plt.yticks(fontsize=18)
plt.xlabel(u'S', fontsize=18)
plt.ylabel(u'S-linegraph size(# of edges)', fontsize=18)
plt.title(u'Diagram of the relationship between S and S-line graph size', fontsize=1
plt.legend(fontsize=18)

plt.show()
```



S-line graph size decreases sharply with increasing S.

S-line graph application(analysis)

In [31]:

```
s1 = s_line_graph_arr[0] # S(=1)-line graph
s3 = s_line_graph_arr[2] # S(=3)-line graph
s5 = s_line_graph_arr[4] # S(=5)-line graph
s7 = s_line_graph_arr[6] # S(=7)-line graph

def find_top_10_betweenness_centrality_score(s):
    betweenness_centrality_score_arr = s.s_betweenness_centrality(normalized=True)
    print('Top #\t\ttitle\t\tbetweenness centrality score')
    title_arr = []
    score_arr = []
    for i in range(10):
        score = max(betweenness_centrality_score_arr)
        hyperedge = np.argmax(betweenness_centrality_score_arr)
        title = hyperedge_to_title_dic[hyperedge]
        score_arr.append(score)
        title_arr.append(title)
        betweenness_centrality_score_arr[hyperedge] = 0
        print('Top ',i + 1, '\t\t', title, '\t\t', score)
    return [title_arr, score_arr]

def find_top_10_closeness_centrality_score(s):
    closeness_centrality_score_arr = s.s_closeness_centrality(v=None)
    print('Top #\t\ttitle\t\tcloseness centrality score')
    title_arr = []
    score_arr = []
    for i in range(10):
        score = max(closeness_centrality_score_arr)
        hyperedge = np.argmax(closeness_centrality_score_arr)
        title = hyperedge_to_title_dic[hyperedge]
        score_arr.append(score)
```

```
        title_arr.append(title)
        closeness_centrality_score_arr[hyperedge] = 0
        print('Top ',i + 1, '\t\t', title, '\t\t', score)
    return [title_arr, score_arr]

def find_top_10_harmonic_closeness_centrality_score(s):
    harmonic_closeness_centrality_score_arr = s.s_harmonic_closeness_centrality(v=No
    print('Top #\t\ttitle\t\tharmonic closeness centrality score')
    title_arr = []
    score_arr = []
    for i in range(10):
        score = max(harmonic_closeness_centrality_score_arr)
        hyperedge = np.argmax(harmonic_closeness_centrality_score_arr)
        title = hyperedge_to_title_dic[hyperedge]
        score_arr.append(score)
        title_arr.append(title)
        harmonic_closeness_centrality_score_arr[hyperedge] = 0
        print('Top ',i + 1, '\t\t', title, '\t\t', score)
    return [title_arr, score_arr]

print('-- Top 10 movies with the highest betweenness centrality score:')
print('\nFor S(=1)-line graph:')
s1_betweenness_result = find_top_10_betweenness_centrality_score(s1)
print('\nFor S(=3)-line graph:')
s3_betweenness_result = find_top_10_betweenness_centrality_score(s3)
print('\nFor S(=5)-line graph:')
s5_betweenness_result = find_top_10_betweenness_centrality_score(s5)
print('\nFor S(=7)-line graph:')
s7_betweenness_result = find_top_10_betweenness_centrality_score(s7)

print('\n\n-- Top 10 movies with the highest closeness centrality score:')
print('\nFor S(=1)-line graph:')
s1_closeness_result = find_top_10_closeness_centrality_score(s1)
print('\nFor S(=3)-line graph:')
s3_closeness_result = find_top_10_closeness_centrality_score(s3)
print('\nFor S(=5)-line graph:')
s5_closeness_result = find_top_10_closeness_centrality_score(s5)
print('\nFor S(=7)-line graph:')
s7_closeness_result = find_top_10_closeness_centrality_score(s7)

print('\n\n-- Top 10 movies with the highest harmonic closeness centrality score:')
print('\nFor S(=1)-line graph:')
s1_harmonic_closeness_result = find_top_10_harmonic_closeness_centrality_score(s1)
print('\nFor S(=3)-line graph:')
s3_harmonic_closeness_result = find_top_10_harmonic_closeness_centrality_score(s3)
print('\nFor S(=5)-line graph:')
s5_harmonic_closeness_result = find_top_10_harmonic_closeness_centrality_score(s5)
print('\nFor S(=7)-line graph:')
s7_harmonic_closeness_result = find_top_10_harmonic_closeness_centrality_score(s7)
```

-- Top 10 movies with the highest betweenness centrality score:

For S(=1)-line graph:

Top #	title	betweenness centrality score
Top 1	Stalingrad	0.09049837291240692
Top 2	Amor de nadie	0.08629073947668076
Top 3	Johnny Casanova and the Case of Secret Serum	0.05841073766350746
Top 4	Grandpa	0.05666143074631691
Top 5	Family of Spies	0.04693305492401123
Top 6	Savage Attraction	0.04259233549237251
Top 7	Play School	0.031361620873212814
Top 8	Joseph McCain/John St. John/Dennis Dutra/Neil Gallagher	0.029640955850481987
Top 9	By Dawn's Early Light	0.02953033335506916
Top 10	Lovely Child	0.027186540886759758

For S(=3)-line graph:

Top #	title	betweenness centrality score
Top 1	For Whom the Bell Klangs: Part 1	2.1961024685879238e-05
Top 2	From Here to Machinery	1.9344686734257266e-05
Top 3	The Adventures of Don Coyote and Sancho Panda	1.77768524736166e-05
Top 4	Tom & Jerry Kids Show	1.2550834981084336e-05
Top 5	The S.S. Drainpipe	1.0455968549649697e-05
Top 6	Tiny Toon Adventures	9.197999133903068e-06
Top 7	The Dream	7.320321401493857e-06
Top 8	It Came from Beneath the Sea Duck	7.292923328350298e-06
Top 9	Splinter Vanishes	6.59489342069719e-06
Top 10	April Gets in Dutch	6.59489342069719e-06

For S(=5)-line graph:

Top #	title	betweenness centrality score
Top 1	Episode #1.6286	5.376502940634964e-06
Top 2	Episode #1.6315	4.907933089270955e-06
Top 3	Episode #1.6387	4.04524507757742e-06
Top 4	Episode #1.6357	3.921283223462524e-06
Top 5	Episode #1.6324	3.062396672248724e-06
Top 6	Episode #1.6188	2.8074609872419387e-06
Top 7	Episode #1.6355	2.673853259693715e-06
Top 8	Episode #1.6351	2.248723149023135e-06
Top 9	Episode #1.6294	2.1847017706022598e-06
Top 10	Episode #1.6288	1.989279326153337e-06

For S(=7)-line graph:

Top #	title	betweenness centrality score
Top 1	Yéti et chuchotement	4.7223039700838854e-08
Top 2	Pif des bois	4.7223039700838854e-08
Top 3	Astynomos Thanasis Papathanasis	1.7052764533787013e-08
Top 4	Voyna na zapadnom napravlenii	1.3117511521443248e-08
Top 5	Buys op de buis	1.1368508801012922e-08
Top 6	Angelos kata lathos	7.870506912865949e-09
Top 7	De stagiaire	5.247004608577299e-09
Top 8	Lies	2.6235023042886496e-09
Top 9	Possessions	2.6235023042886496e-09
Top 10	Pretenders	2.6235023042886496e-09

-- Top 10 movies with the highest closeness centrality score:

For S(=1)-line graph:

```
Top #           title           closeness centrality score
Top  1           Domo Arigato            1.0
Top  2           Spy Story               1.0
Top  3           Halfaouine: Boy of the Terraces                 1.0
Top  4           Michigan mélodie                1.0
Top  5           Astonished              1.0
Top  6           Dragon Hunt             1.0
Top  7           Horrorshow              1.0
Top  8           Slaughter in Xian               1.0
Top  9           The Figures from Earth                  1.0
Top  10                 Demonia                 1.0

For S(=3)-line graph:
Top #           title           closeness centrality score
Top  1           Born in '45             nan
Top  2           Die Kupferfalle                 1.0
Top  3           Wenn du groß bist, lieber Adam                  1.0
Top  4           Neuner                  1.0
Top  5           Lauf eines Todes                1.0
Top  6           Zeitzünder              1.0
Top  7           Die Freiheit der Kunst                  1.0
Top  8           Jede Menge Abschied             1.0
Top  9           Blumen für den Rechtsanwalt             1.0
Top  10                 Anwälte unter sich              1.0

For S(=5)-line graph:
Top #           title           closeness centrality score
Top  1           Born in '45             nan
Top  2           Die Kupferfalle                 1.0
Top  3           Wenn du groß bist, lieber Adam                  1.0
Top  4           Neuner                  1.0
Top  5           Lauf eines Todes                1.0
Top  6           Zeitzünder              1.0
Top  7           Die Freiheit der Kunst                  1.0
Top  8           Jede Menge Abschied             1.0
Top  9           Blumen für den Rechtsanwalt             1.0
Top  10                 Anwälte unter sich              1.0

For S(=7)-line graph:
Top #           title           closeness centrality score
Top  1           Born in '45             nan
Top  2           Die Kupferfalle                 1.0
Top  3           Wenn du groß bist, lieber Adam                  1.0
Top  4           Neuner                  1.0
Top  5           Lauf eines Todes                1.0
Top  6           Zeitzünder              1.0
Top  7           Die Freiheit der Kunst                  1.0
Top  8           Jede Menge Abschied             1.0
Top  9           Blumen für den Rechtsanwalt             1.0
Top  10                 Anwälte unter sich              1.0


-- Top 10 movies with the highest harmonic closeness centrality score:

For S(=1)-line graph:
Top #           title           harmonic closeness centrality score
Top  1           TaleSpin                3199.692626953125
Top  2           From Here to Machinery                  3155.24755859375
Top  3           Tiny Toon Adventures            3153.000244140625
Top  4           A Bad Reflection on You: Part 2                 3151.518798828125
Top  5           Plunder and Lightning: Part 4           3140.621826171875
Top  6           A Bad Reflection on You: Part 1                 3130.05078125
Top  7           Plunder and Lightning: Part 2           3130.05078125
Top  8           Polly Wants a Treasure                  3127.771240234375
```

```
Top  9           Plunder and Lightning: Part 1           3124.712646484375
Top  10                 Midnight Patrol: Adventures in the Dream Zone          312
3.3095703125

For S(=3)-line graph:
Top #           title           harmonic closeness centrality score
Top  1           Episode #1.6397                 217.0
Top  2           Episode #1.6301                 207.16665649414062
Top  3           Episode #1.6201                 204.8333282470703
Top  4           Episode #1.6357                 204.16665649414062
Top  5           Episode #1.6324                 204.0
Top  6           Episode #1.6315                 203.8333282470703
Top  7           Episode #1.6387                 203.8333282470703
Top  8           Episode #1.6195                 202.16665649414062
Top  9           Episode #1.6220                 201.3333282470703
Top  10                 Episode #1.6185                 200.3333282470703

For S(=5)-line graph:
Top #           title           harmonic closeness centrality score
Top  1           Episode #1.6315                 78.85004425048828
Top  2           Episode #1.6324                 78.07621002197266
Top  3           Episode #1.6387                 77.10001373291016
Top  4           Episode #1.6351                 76.71666717529297
Top  5           Episode #1.6188                 76.61666870117188
Top  6           Episode #1.6288                 75.80001068115234
Top  7           Episode #1.6286                 74.04998016357422
Top  8           Episode #1.6355                 73.70000457763672
Top  9           Episode #1.6357                 72.76665496826172
Top  10                 Episode #1.6224                 72.53333282470703

For S(=7)-line graph:
Top #           title           harmonic closeness centrality score
Top  1           Yéti et chuchotement            15.0
Top  2           Pif des bois            15.0
Top  3           Cancres et chouchoux            13.5
Top  4           Le blues sur le lagon           13.5
Top  5           Une partie de campagne                  13.5
Top  6           Gare aux fantômes               13.5
Top  7           Aïe mon neveu           13.5
Top  8           Casse à tous les étages                 13.5
Top  9           Ballade pour une valise                 13.5
Top  10                 Chaperon rouge et compagnie             13.5
```

Diagrams of Top 10 movies based on different scores(S=1/3/5/7)

In [32]:
```
def plot_diagram_of_top_10_score(result, output):
    x = result[0]
    y = result[1]
    plt.figure(figsize=(16, 6))
    plt.bar(x, y, width=0.5)
    plt.xticks(fontsize=10, rotation=20)
    plt.yticks(fontsize=18)
    plt.ylabel(output, fontsize=20)
    diagram_title = 'Diagram of Top 10 movies with the highest ' + output
    plt.title(diagram_title, fontsize=20)
    plt.show()
```

Diagram of Top 10 movies with the highest betweenness centrality score(S=1)

In [33]:
```
plot_diagram_of_top_10_score(s1_betweenness_result, 'betweenness centrality score(S=
```

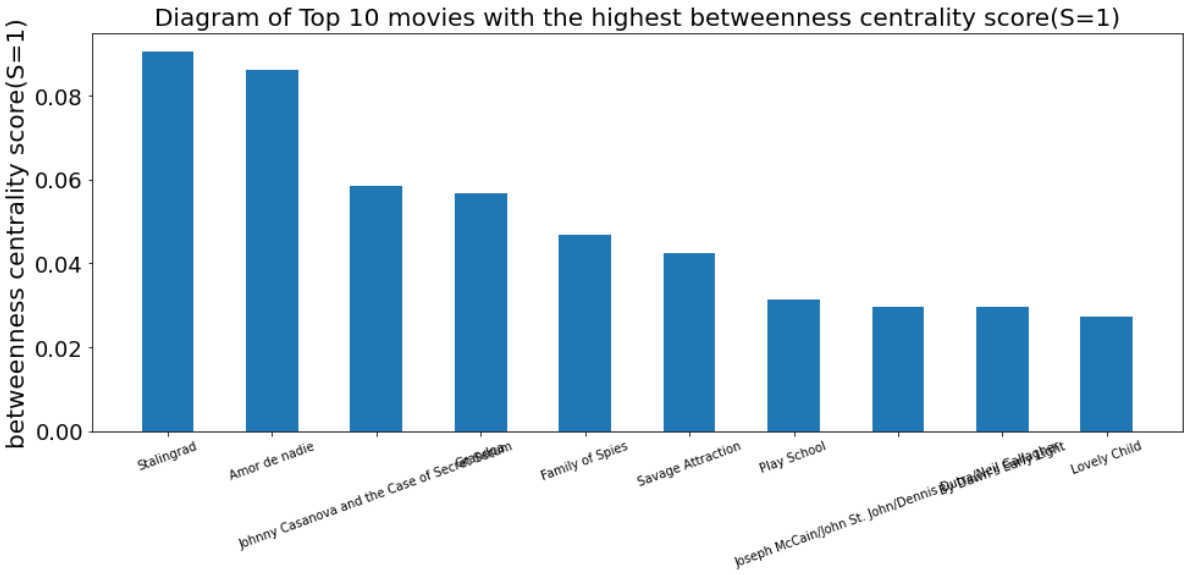


Diagram of Top 10 movies with the highest betweenness centrality score(S=3)

In [34]:
```
plot_diagram_of_top_10_score(s3_betweenness_result, 'betweenness centrality score(S=
```

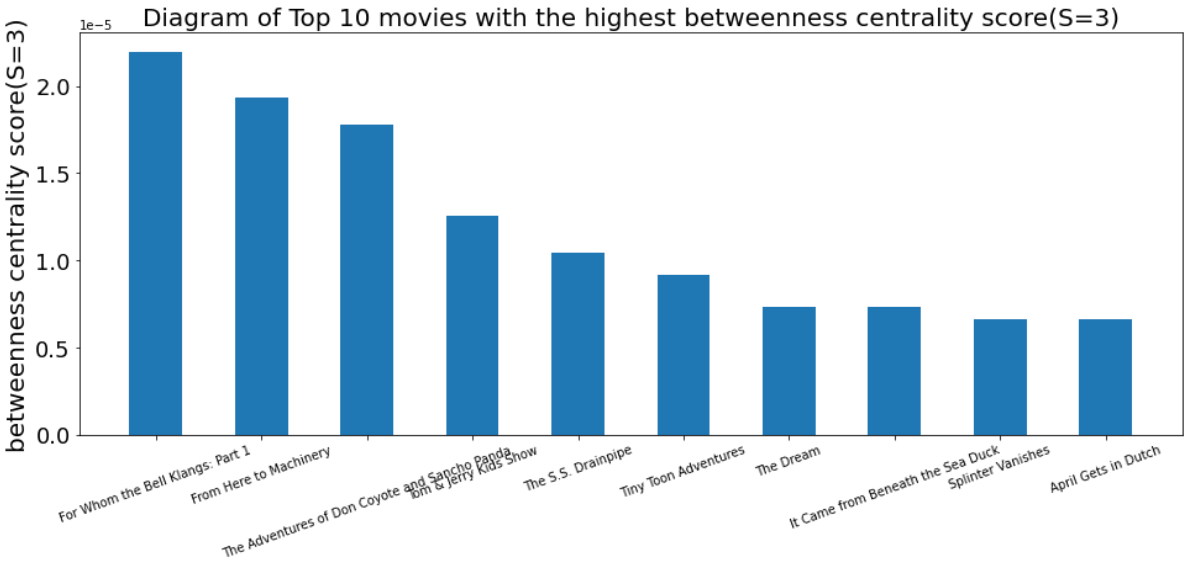


Diagram of Top 10 movies with the highest betweenness centrality score(S=5)

In [35]:
```
plot_diagram_of_top_10_score(s5_betweenness_result, 'betweenness centrality score(S=
```

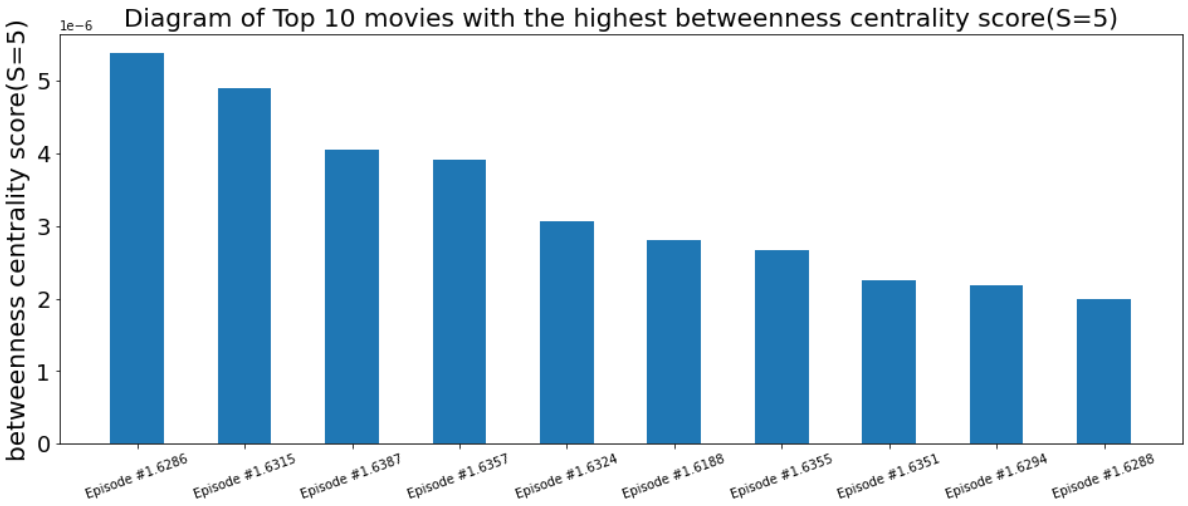


Diagram of Top 10 movies with the highest betweenness centrality score(S=7)

In [36]:
```
plot_diagram_of_top_10_score(s7_betweenness_result, 'betweenness centrality score(S=
```

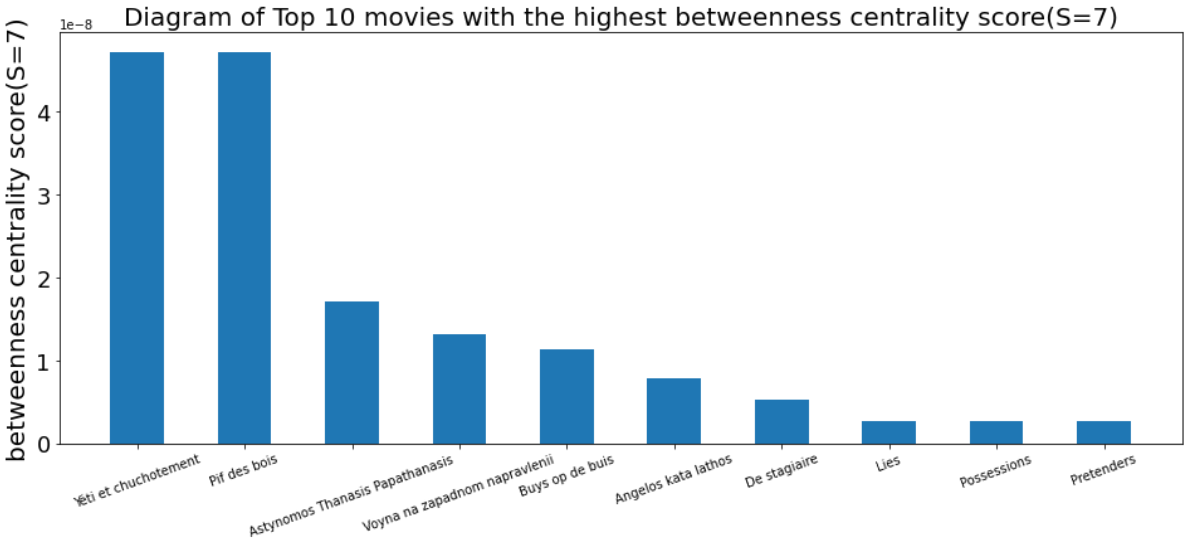


Diagram of Top 10 movies with the highest closeness centrality score(S=1)

In [37]:
```
plot_diagram_of_top_10_score(s1_closeness_result, 'closeness centrality score(S=1)')
```

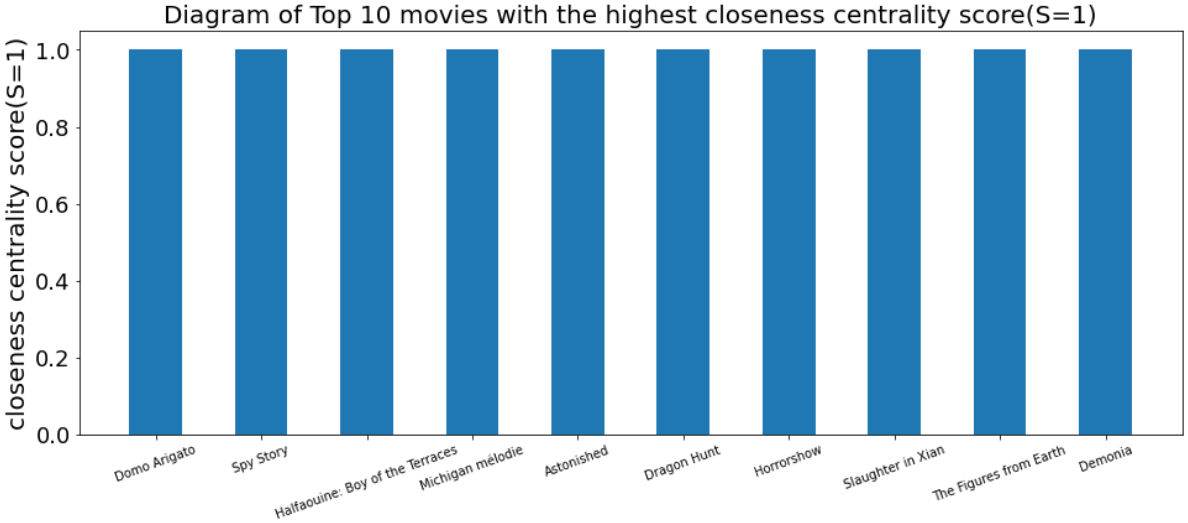


Diagram of Top 10 movies with the highest closeness centrality score(S=3)

In [38]:
```
plot_diagram_of_top_10_score(s3_closeness_result, 'closeness centrality score(S=3)')
```

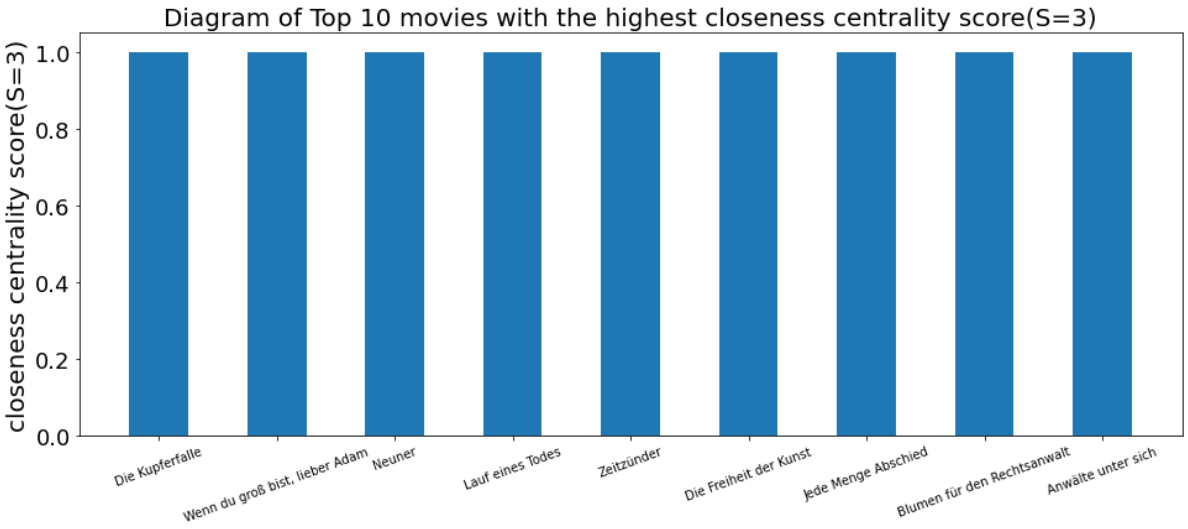


Diagram of Top 10 movies with the highest closeness centrality score(S=5)

In [39]:
```
plot_diagram_of_top_10_score(s5_closeness_result, 'closeness centrality score(S=5)')
```

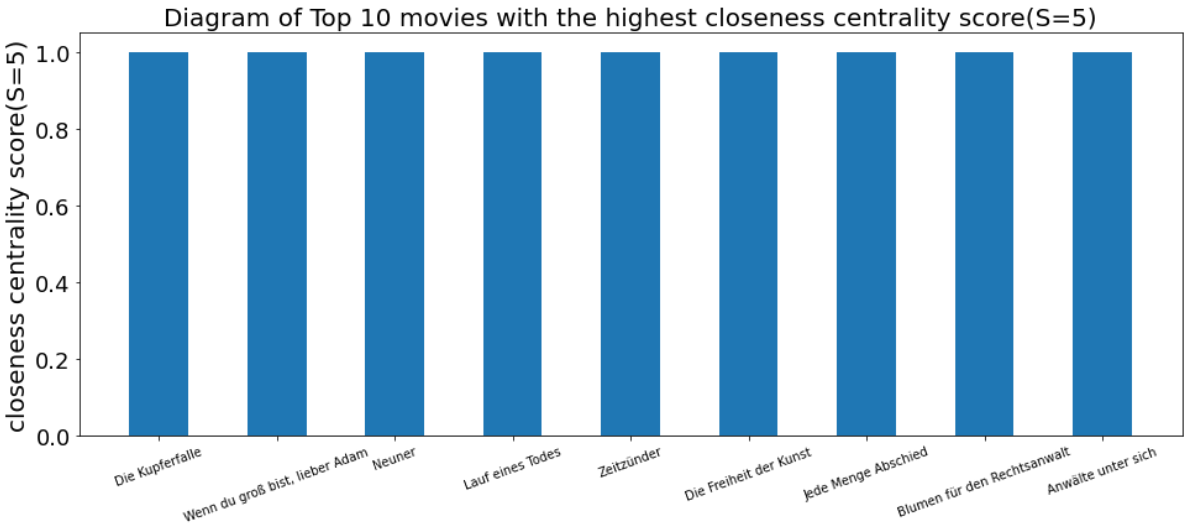


Diagram of Top 10 movies with the highest closeness centrality score(S=7)

In [40]:
```
plot_diagram_of_top_10_score(s7_closeness_result, 'closeness centrality score(S=7)')
```

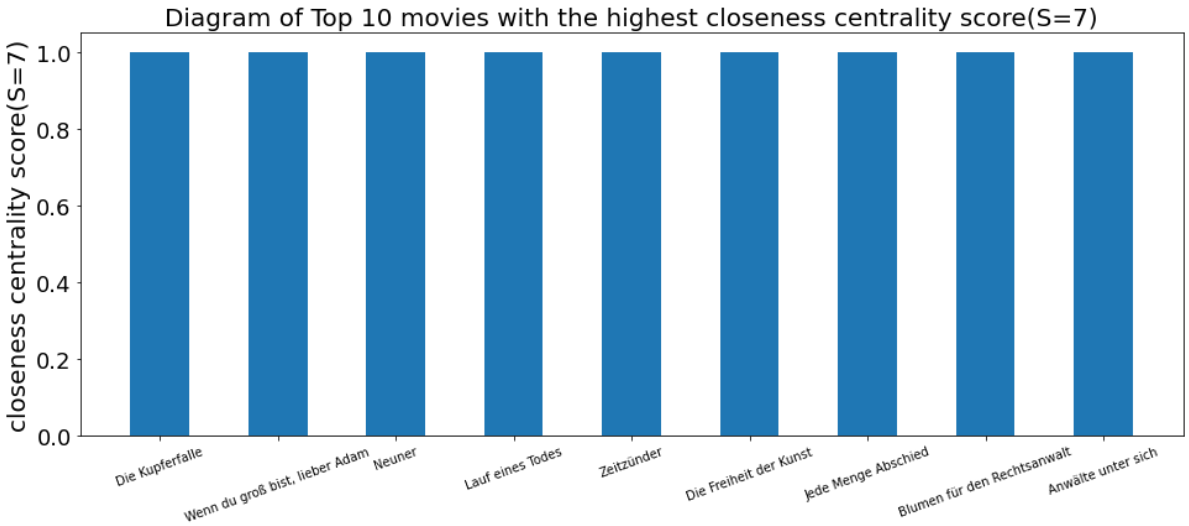


Diagram of Top 10 movies with the highest harmonic closeness centrality score(S=1)

In [41]:
```
plot_diagram_of_top_10_score(s1_harmonic_closeness_result, 'harmonic closeness centr
```

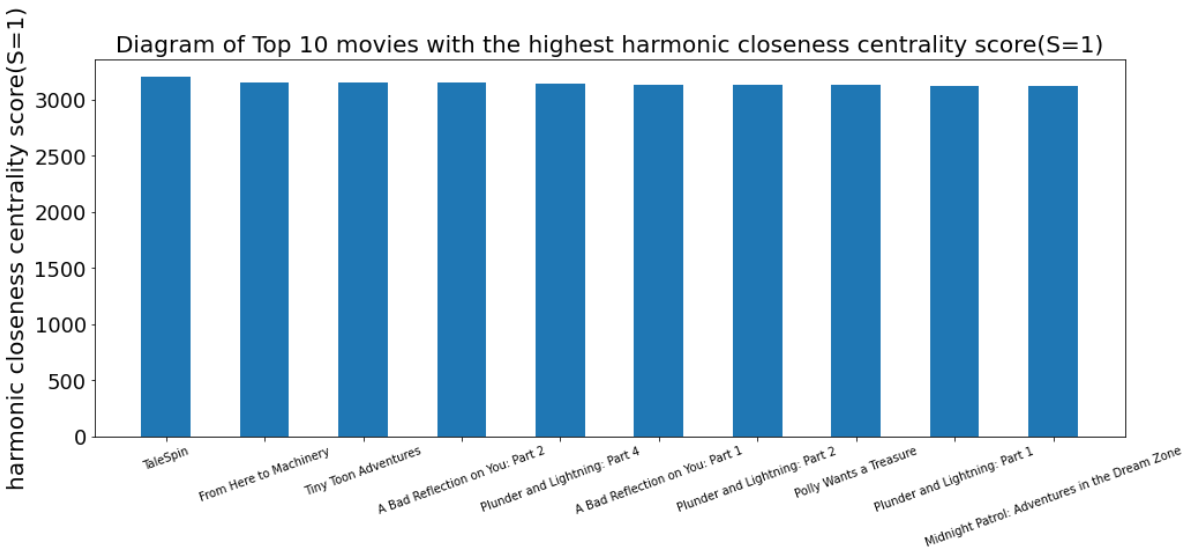


Diagram of Top 10 movies with the highest harmonic closeness centrality score(S=3)

In [42]:
```
plot_diagram_of_top_10_score(s3_harmonic_closeness_result, 'harmonic closeness centr
```

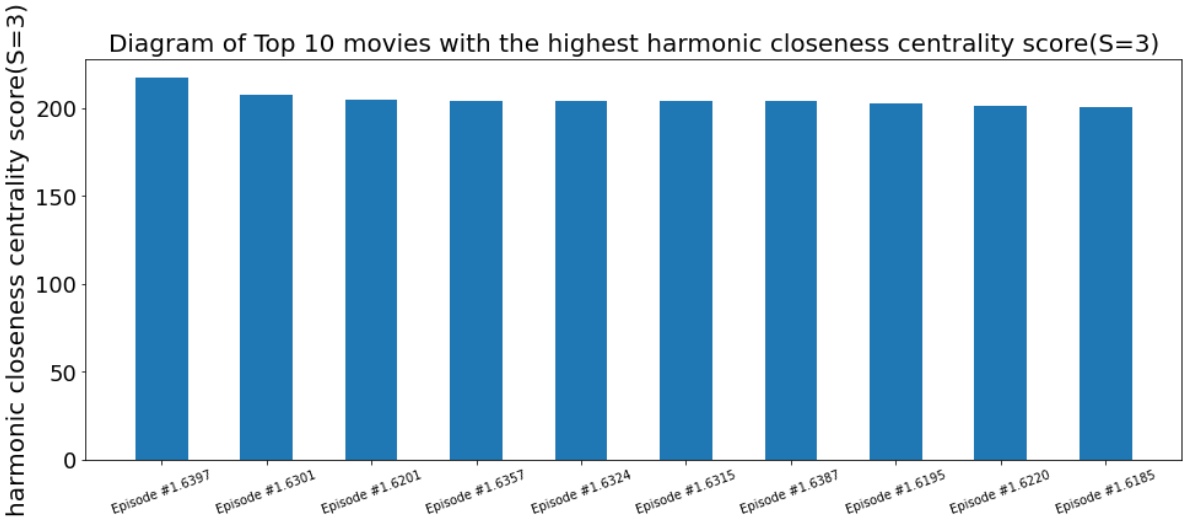


Diagram of Top 10 movies with the highest harmonic closeness centrality score(S=5)

In [43]:
```
plot_diagram_of_top_10_score(s5_harmonic_closeness_result, 'harmonic closeness centr
```

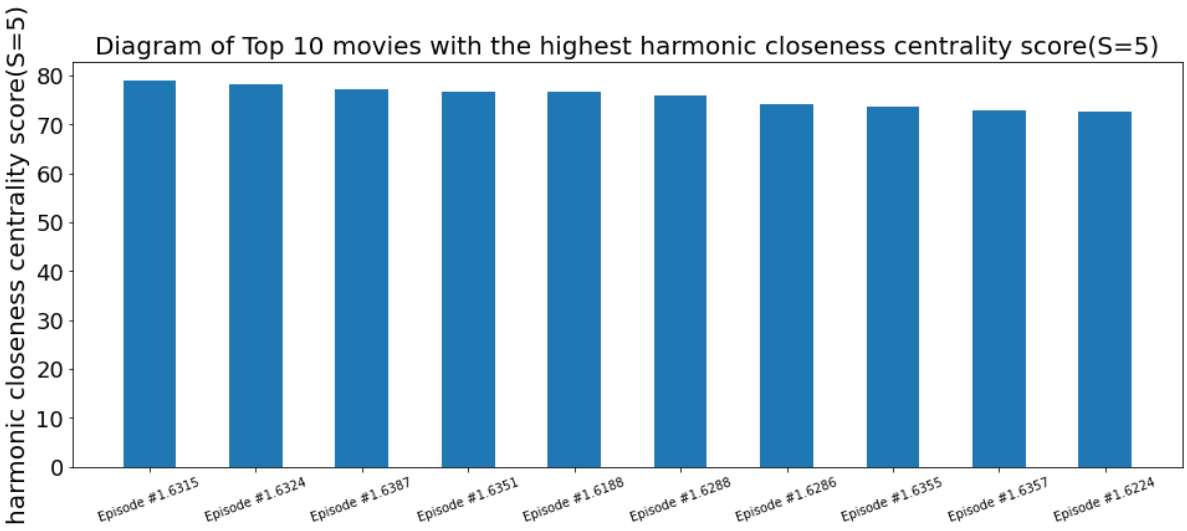


Diagram of Top 10 movies with the highest harmonic closeness centrality score(S=7)

In [44]:
```
plot_diagram_of_top_10_score(s7_harmonic_closeness_result, 'harmonic closeness centr
```

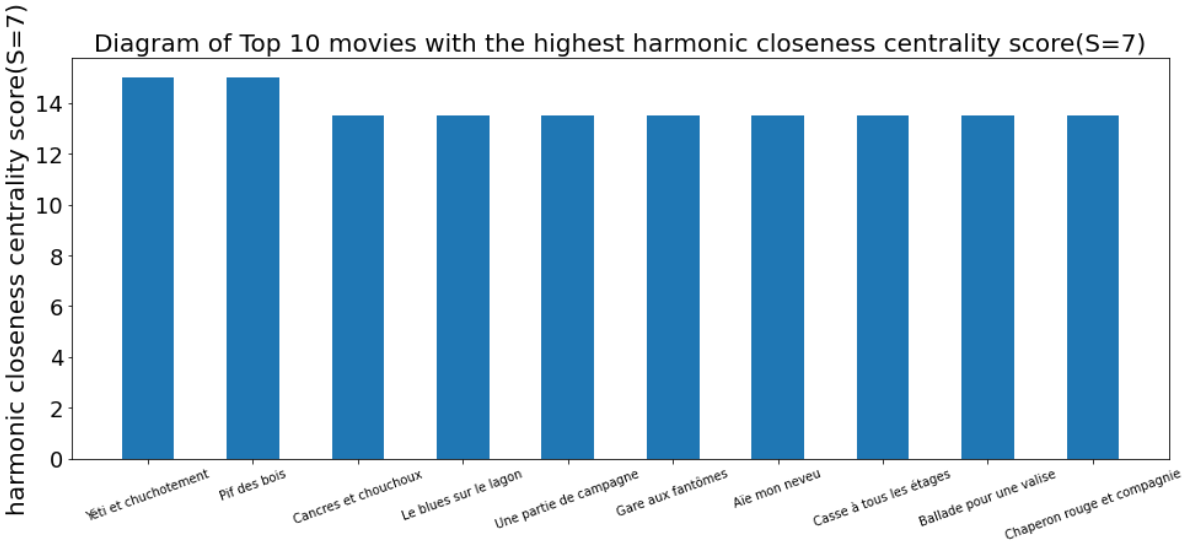


Diagram of Top 10 movies with the highest harmonic closeness centrality score(S=7)
harmonic closeness centrality score(S=7)
0
2
4
6
8
10
12
14
Yéti et chuchotement
Pif des bois
Cancres et chouchoux
Le blues sur le lagon
Une partie de campagne
Gare aux fantômes
Aïe mon neveu
Casse à tous les étages
Ballade pour une valise
Chaperon rouge et compagnie