PROJECT PRESENTATION

Created by Francesco Villi

TASK

From a dataset of movies extract all data, create a graph using Networkx, and resolve the exercises written in the next slides.

Before starting we need to create the graph...

CREATION GRAPH

CREATION GRAPH

- Split values actor and movie
- Check if the file is ended
- Detect film, year and actor
- Create or retrieve the key for film and actor

```
file=open('imdb-actors-actresses-movies.tsv','r')
EOF=False
id_actor=0
id_film=1
count_actor=0
count_film=1
actor_to_key={}
key_to_actor={}
film_to_key={}
key_to_film={}
G=nx.Graph()
while not EOF:
  line=file.readline().split('\n')
  record=line[0].split('\t')
  if  len(record)<2:
```

RESULT

- The graph has 3.110.737 nodes where 2.364.796 are actors and 745.941 are movies
- The amount of edges is 8103960
- The algorithm showed in the previous slide takes about 1 minute to be executed

EXERCISE 1

Which is the movie with the largest number of actors, considering only the movies up to year x= {1930,1940,1950,1960,1970,1980,1990,2000,2010,2020}?

ALGORITHM

- For each film, check the year
- Calculate the number of actors
- Update the best result

```
x = [1930,1940,1950,1960,1970,1980,1990,2000,2010,2020]
up_to_year=x[rnd.randint(0,len(x)-1)]
best_result={'count':0,'movies':[]}
for film,name_year in key_to_film.items():
  if name_year[1]<=up_to_year:
    tot_movie=len(G[film])
    if(tot_movie>=best_result['count']):
      if tot_movie>best_result['count']:
        best_result['movies']=[]
        best_result['count']=tot_movie
      best_result['movies'].append(film)
```

RESULTS

Year	Film	Nr actors
1930	The King of Kings (1927)	171
1940	The Buccaneer (1938)	219
1950	Gone to Earth (1950)	290
1960	Around the World in Eighty Days (1956)	1298
1970	Around the World in Eighty Days (1956)	1298
1980	Around the World in Eighty Days (1956)	1298
1990	Around the World in Eighty Days (1956)	1298
2000	Around the World in Eighty Days (1956)	1298
2010	Around the World in Eighty Days (1956)	1298
2020	Around the World in Eighty Days (1956)	1298

EXERCISE 2

Considering only the movies up to year x = {1930,1940,1950,1960,1970,1980,1990,2000,2010,2020} and restricting to the largest connected component of the graph. Compute exactly the diameter of G

This Algorithm is based on 3 steps:

- Method for finding the largest connected component H according to the film's year
- two_sweep for detecting the starting node in the subgraph H
- iFUB for computing the exact diameter of H

TWO_SWEEP

```
@py_random_state(1)
def two_sweep(Graph, seed):
  rnd_node = seed.choice(list(Graph))
  source= list(nx.single_source_shortest_path_length(Graph, rnd_node))[-1]
  distances_b = list(nx.single_source_shortest_path_length(Graph, source).items())
  index_start_node=bisect.bisect_left(distances_b,int(distances_b[-1][1]/2),key=lambda k:k[1])
  index_end_node=bisect.bisect_right(distances_b,int(distances_b[-1][1]/2),key=lambda k:k[1])
  return max(distances_b[index_start_node::index_end_node],key=lambda k:len(H_small[k[0]]))
```

- Select a random source node and get the farthest node from the source
- Compute the shortest path lengths from source to all reachable nodes.
- Select a node in the middle with the highest degree

iFUB

```
def iFUB(G,node_start):
  list_fringe=itertools.groupby(reversed(nx.single_source_shortest_path_length(G,node_start).items()),
            key=lambda k:k[1])
  group=next(list_fringe)
  i=group[0]
  lb=i
  ub=2*i
  max_ecc=0
```

- Get a list of nodes from the starting node
- Set a lower and upper bound
- For each node at distance i, compute eccentricity and check if it's equal or greater than upper bound
- If the previous condition isn't triggered, update the lower and upper bound

ALGORITHM

```
x = [1930,1940,1950,1960,1970,1980,1990,2000,2010,2020]
year=x[rnd.sample(x,1)]
list_nodes=set()
for u,v in G.edges():
  if G[u][v]['year'] <= year:
    list_nodes.add(u)
    list_nodes.add(v)
largest_cc=max(nx.connected_components(G.subgraph(list_nodes)),key=len)
H_small=G.subgraph(largest_cc)
node_start=two_sweep(H_small,None)
iFUB(H_small,node_start[0])
```

- Find all nodes that respect the year's constraint
- Find maximum connected component and create a SubGraph view of the subgraph induced on nodes
- Compute the diameter of the subgraph

Observations

- In some cases, the computing of the diameter can take much time due to the unlucky starting node

RESULTS

Year	Diameter	Time	Nr BFS
1930	32	58 sec	43
1940	36	52 sec	20
1950	38	87 sec	35
1960	34	57 sec	11
1970	25	4 min	25
1980	25	5 min	25
1990	30	6 min	48
2000	28	11 min	64
2010	30	2 h	295
2020	32	15 min	28

EXERCISE 3

Which is the movie with the largest number of popular actors, i.e. such that the sum of the number of movies its actors participated in is maximum?

ALGORITHM

- For each film retrieve the list of actors
- For each actor in that list, compute the popularity
- Compare each sum with the best result

```
largest_movie={'count':0,'movies':[]}
for film,_ in key_to_film:
  edges=G.edges(film)
  count=sum(G.degree(act) for flm,act in G.edges(film))
  if count>=largest_movie['count']:
    if count>largest_movie['count']:
      largest_movie['count']=count
      largest_movie['movies']=[]
    largest_movie['movies'].append(film)
print(largest_movie)
```

RESULTS

Film	Popularity
MILF Madness (2012) (V)	34181

EXERCISE 4

Build also the actor graph, whose nodes are only actors and two actors are connected if they did a movie together, which is the pair of actors who collaborated the most among themselves?

CREATE ACTOR GRAPH

```
def create_graph_actors_partilally_ordered():
  collaborations={}
  best_collaborators=[0,[]]
  G_actor=nx.Graph()
  for actor_1 in key_to_actor:
    list_films=itertools.chain(G[actor_1])
    for film in list_films:
      list_actor_for_film=itertools.chain(G[film])
      for actor_2 in list_actor_for_film:
        if actor_2>actor_1:   # avoid duplication
          if actor_2 in collaborations:
```

- For each actor get the list of films in which he partecipated
- For each film, get the list of actor in that film
- Compute the collaborations between actors maintaining the greatest
- Add best and other collaborations to the graph

ALGORITHM

```
best_result={'count':0,'pairs':[]}
for actor in itertools.chain(G_actor):
  for actor_2 in itertools.chain(((G_actor[actor]))):
    if actor_2>actor:
      colls=G_actor[actor][actor_2]['weight']
      if colls>=best_result['count']:
        if colls>best_result['count']:
          best_result['count']=colls
```

- For each actor, get the view of adjacent nodes
- Get the best weighed edge not measured until now
- We start to compare the best collaboration of that actor with the best result
- If the count of collaborators is lower than the best result, we skip to the next actor

RESULTS

Actors	Collaborations
Byron, Tom (I) - North, Peter (I)	420

Algorithm	Time
create_graph_actors_partially_ordered	6 m
find_best_collaborations	4 m