Mancini

November 19, 2023

1 Progetto di fine corso Algoritmi e programmazione per l'analisi dei dati 2022-2023

Il progetto prevede di rispondere a delle domande che verranno enunciate in seguito, analizzando un insieme di datests contenenti le informazioni sulle pubblicazioni scientifiche. Lo scopo è quello di effettuare l'analisi utilizzando algoritmi sui grafi. La libreria utilizzata per la costruzioni e metodi di utilità di grafi è NetworkX.

1.1 Operazioni preliminari

Prima di iniziare l'alalisi, importiamo i principali moduli che usere:

- NetworkX
- Pandas
- Numpy

Queste sono le librerie più usate in ambito di analisi dei dati.

[1]:
```
import networkx as nx
import pandas as pd
import numpy as np
```

Essendo i dataset dei file csv, li cerchiamo all'interno della cartella del progetto, più precisamente ci aspettiamo che sia in una cartella chiamata `Data`.

Utilizzando il metodo `os.listdir(nome_path)` del modulo integrato di Python `os`, possiamo ottenere una lista con i nomi di tutti i file in quella cartella.

> Simile al comando shell `ls`.

[2]:
```
import os

path = "Data"
csv_list = os.listdir(path)
```

A questo punto possiamo leggere i csv. La lettura è eseguita utilizzando la libreria `pandas`. Essa offre una conveniente struttura dati per la rappresentazione di un dataset che prende il nome di `DataFrame`. Può essere visto come una matrice, implementata tramite lista di liste.

Una lettura molto semplice richiede solo la specifica del nome del csv da leggere e il metodo `read_csv` restituirà un `DataFrame`.

Nel codice sottostante è stato eseguito un wrapper del metodo sopra citato. Sappiamo in anticipo che colonne leggere, fatta eccezione di un unico file `out-dblp_proceedings.csv`, che ha una colonna diversa, ma con lo stesso significato al fine della nostra analisi.

L'operazione aggiuntiva è stata quella di dare un nome al Dataframe (che sarà comodo più avanti).

[3]:
```
def read_csv(name:str)->pd.DataFrame:
    cols = ["id","author","title"]
    if name == "out-dblp_proceedings.csv":
        cols = ["id","editor","title"]

    df = pd.read_csv(
    path +"/" + name,
    delimiter=";",
    usecols=cols,
    nrows=5000
    )
    df.name = name.split(".")[0]
    df.rename(columns={"editor":"author"}, inplace=True)
    return df
```

1.2 Creazione della lista dei DataFrame

Una volta che abbiamo una funzione per leggere i csv nel modo che desideriamo, ciò che vogliamo ottenere è una lista di dataframe da poter utilizzare in seguito.

Questo problema è facilmente risolvibile con un ciclo. In aggiunta, essendoci dei valori nulli, li togliamo a lettutra ultimata.

[4]:
```
df_list = list()
for csv in csv_list:
    df_list.append(
        read_csv(csv)
    )
    df_list[-1].dropna(inplace = True)
```

1.2.1 Esempio di DataFrame

Di seguito vengono riportate le prime 5 righe del primo DataFrame creato.

[5]:
```
print(f"Name: {df_list[0].name}")
df_list[0].head()
```

```
Name: out-dblp_article
```

[5]:

	id	author	\
3	4105295	Clement T. Yu|Hai He|Weiyi Meng|Yiyao Lu|Zongh…	
4	4106479	Fatih Gelgi|Hasan Davulcu|Srinivas Vadrevu	
5	4107897	Daniel A. Menascé|Vasudeva Akula	
6	4108498	Hongjian Fan|Kotagiri Ramamohanarao	

```
7  4108959                          Bing Liu 0001|Yanhong Zhai

                                                      title
3  Towards Deeper Understanding of the Search Int…
4  Information Extraction from Web Pages Using Pr…
5  Improving the Performance of Online Auctions T…
6                         Patterns Based Classifiers.
7  Extracting Web Data Using Instance-Based Learn…
```

1.3 Creazione dei Grafi

Passiamo ora alla creazione del `Grafo`. Vogliamo costruire un grafo bipartito, contenente in un insieme gli `autori` e nell' altro i `titoli`.

Ciò che dobbiamo fare è dividere i vari autori di una singola pubblicazione e creare i nodi e archi desiderati.

> Gli autori sono separati da carattere `|`.

Per comodità come attributo di ogni nodo degli autori, inseriamo il conteggio degli autori che hanno partecipato a tale pubblicazione. (Questo passaggio può essere evitato perchè corrisponde al grado di tale nodo).

Appendiamo in una lista i vari grafi che abbiamo creato.

[6]:
```python
def create_graph(df:pd.DataFrame)->nx.Graph:
    G = nx.Graph()
    for publication_id, row in df.iterrows():
        authors = row["author"].split("|")
        title = row["title"]
        G.add_node(publication_id, bipartite = 0, title=title, authors_counter␣
 ↪= len(authors))
        for author in authors:
            G.add_node(author, bipartite = 1)
            G.add_edge(publication_id,author)
    return G

graph_list = list()
for df in df_list:
    graph_list.append(
        create_graph(df)
    )
```

1.3.1 Note su una possibile implementazione parallela.

La creazione del grafo di NetworkX avviene in maniera sequenziale ed è chiaramente un'operazione non vettorizzabile. Tuttavia sapendo in anticipo (come nel nostro caso) il numero di grafi da costruire, fornendo una struttura dati di output con la stessa dimensione del numero di grafi da costruire possiamo eseguire in parallelo sulla CPU la creazione di tali grafi. Naturalmente vanno fatte considerazioni aggiuntive sull'utilizzo della memoria.

La libreria più semplice da usare per gestire il parallelismo sulla CPU (senza dover instanziare altri interpreti) è probabilmente Numba, che ci permette di avere un compilatore *just in time*.

1.4 Pubblicazione con maggior numero di autori

Per trovare la pubblicazione con il maggior numero di autori, dobbiamo visitare il grafo nel seguente modo: 1. Selezionare tutti i nodi delle pubblicazioni 2. All'interno di questi nodi è presente il campo `author_counter`. 3. Castando la lista ad array di `numpy`, possiamo utilizzare `argmax`. 4. Avendo l'id della pubblicazione selezioniamo il campo `titolo`.

> Se non avessimo avuto il campo `author_counter`, avremmo potuto contare il grado del nodo.

[7]:
```
def get_publication_with_max_authors(G:nx.Graph)->tuple[str,int]:
    publication_ids = list(n for n, d in G.nodes(data=True) if d["bipartite"]␣
 ↪== 0)

    authors_counter_array = np.array(
        list(map(lambda id: G.nodes[id]["authors_counter"], publication_ids))
        )

    max_authors_pubication_id = publication_ids[authors_counter_array.argmax()]

    title = G.nodes[max_authors_pubication_id]["title"]

    return (
        title,
        G.nodes[max_authors_pubication_id]['authors_counter']
    )

for idx, G in enumerate(graph_list):
    title, counter = get_publication_with_max_authors(G)
    print(f"-------------Graph: {df_list[idx].name}--------------")
    print(f"{title}\nWith {counter} authors.\n")
```

```
-------------Graph: out-dblp_article--------------
Making Bertha Drive - An Autonomous Journey on a Historic Route.
With 31 authors.

-------------Graph: out-dblp_book--------------
The Munich Project CIP, Volume I: The Wide Spectrum Language CIP-L
With 18 authors.

-------------Graph: out-dblp_incollection--------------
CoolEmAll: Models and Tools for Planning and Operating Energy Efficient Data
Centres.
With 19 authors.

-------------Graph: out-dblp_inproceedings--------------
```

```
R-GMA: An Information Integration System for Grid Monitoring.
With 21 authors.

-------------Graph: out-dblp_mastersthesis--------------
Shadow Paging Is Feasible.
With 1 authors.

-------------Graph: out-dblp_phdthesis--------------
Datenbankgestütztes Kosten- und Erlöscontrolling: Konzept und Realisierung einer
entscheidungsorienerten Erfolgsrechnung.
With 2 authors.

-------------Graph: out-dblp_proceedings--------------
Workshop Proceedings of the 8th International Conference on Intelligent
Environments, Guanajuato, Mexico, June 26-29, 2012
With 22 authors.
```

1.5 Autore con maggior numero di collaborazioni

Per trovare l'autore con il maggior numero di collaboratori dobbiamo:

1. Selezionare tutti i nodi degli autori.
2. Per ognuno degli autori selezioniore le proprie pubblicazioni.
3. Per ogni pubblicazione salvarsi i collaboratori (ad esempio in una lista).
4. La lista più grande (una per ogni autore) corrisponde alla risposta della domanda.

[8]:
```python
def get_author_with_most_collaborations(G:nx.Graph)->tuple[str,int]:
    authors = list(n for n, d in G.nodes(data=True) if d["bipartite"] == 1)

    max = {"author": "None","collaborators":list(),"count":0}
    for author in authors:
        collaborators = list()
        publication_ids = [publication_id[1] for publication_id in list(G.
 ↪edges(author))]
        for publication_id in publication_ids:
            collaborators.append(
                [publication_id[1] for publication_id in list(G.
 ↪edges(publication_id))]
            )
        if len(collaborators)  > max["count"]:
            max["author"] = author
            max["count"] = len(collaborators)
    return(
        max['author'],
        max["count"]
    )
```

```
for idx, G in enumerate(graph_list):
    author, count= get_author_with_most_collaborations(G)
    print(f"-------------Graph: {df_list[idx].name}--------------")
    print(f"{author}, with {count} collaborators.\n")
```

```
-------------Graph: out-dblp_article--------------
Edmond Bianco, with 169 collaborators.

-------------Graph: out-dblp_book--------------
Bertrand Meyer 0001, with 15 collaborators.

-------------Graph: out-dblp_incollection--------------
Christian S. Jensen, with 79 collaborators.

-------------Graph: out-dblp_inproceedings--------------
Ali Özen, with 28 collaborators.

-------------Graph: out-dblp_mastersthesis--------------
Tatu Ylönen, with 1 collaborators.

-------------Graph: out-dblp_phdthesis--------------
Alberto Bonanno, with 2 collaborators.

-------------Graph: out-dblp_proceedings--------------
Joaquim Filipe, with 83 collaborators.
```

1.6 Calcolo del Diametro

Per il calcolo del Diametro utiliziamo l'algoritmo `iFub` visto a lezione.

Dobbiamo prima implementare delle funzioni di utilità quali: - Calcolo della componente fortemente connessa più grande (il grafo risultante non è connesso e quindi l'algoritmo non funzionerebbe). - Scelta del nodo iniziale - `two-sweep`

Per il calcolo del nodo iniziale abbiamo due opzioni, selezionabili durante la chiamata del metodo: 1. Nodo casuale 2. Nodo *mediano* nel cammino del **two-sweep**.

[9]:
```
def get_largest_connected_component(G:nx.Graph)->nx.Graph:
    return G.subgraph(
    sorted(nx.connected_components(G), key = len, reverse=True)[0]
    ).copy()

def find_farther_node(G,starting_node:str)->list:
    edges = nx.bfs_edges(G,starting_node)
    edges = [starting_node] + [v for u, v in edges]
    return list(G.edges(edges[-1]))[0][0]

def two_sweep_path(G:nx.Graph,starting_node:str)->list:
```

```
    a = find_farther_node(G,starting_node)
    b = find_farther_node(G,a)
    return nx.shortest_path(G,a,b)

from random import choice
def calculate_starting_node(G: nx.Graph, method: str = "random") -> str:
    random_node = choice(list(G.nodes))
    if method == "random":
        return random_node
    elif method == "2-sweep":
        starting_node = two_sweep_path(G, random_node)
        median_idx = len(starting_node) // 2
        starting_node = starting_node[median_idx]
    else:
        raise ValueError("Metodo non valido. Usare 'random' o '2-sweep'.")

    return starting_node
```

Per il calcolo del diametro con **iFub** dobbiamo anche implementare le funzioni per il calcolo dell'insieme $F_i(u)$ e $B_i(u)$, con u nodo iniziale.

Il diametro calcolato nel seguente modo restituisce (nei test effettuati) lo stesso numero del metodo `diameter()` di NetwokX.

[10]:
```
def calculate_F(G:nx.Graph,node:str,distance:int)->set:
    return nx.descendants_at_distance(G,node,distance)

def calculate_B_i(G:nx.Graph, node:str, i:int)->int:
    F = calculate_F(G,node,distance=i)
    B_i = 0
    for node in F:
        max = nx.eccentricity(G, v=node)
        if max > B_i:
            B_i = max
    return B_i
```

[11]:
```
def iFub(G:nx.Graph, start_method:str = "random")-> int:
    G = get_largest_connected_component(G)
    node = calculate_starting_node(G,method=start_method)
    i = nx.eccentricity(G,v=node)

    lb = i
    ub = 2*lb

    while ub > lb:
        B_i = calculate_B_i(G,node,i)
        max = np.max([lb,B_i])
        if max > 2*(i-1):
```

```
            return max
        else:
            lb = max
            ub = 2*(i-1)
        i=i-1
    return lb

for idx, G in enumerate(graph_list):
    diameter = iFub(G)
    print(f"Graph {df_list[idx].name} has diameter: {diameter}")
    print(f"NetworkX diameter is: {nx.
 ↪diameter(get_largest_connected_component(G))}\n")
```

```
Graph out-dblp_article has diameter: 52
NetworkX diameter is: 52

Graph out-dblp_book has diameter: 6
NetworkX diameter is: 6

Graph out-dblp_incollection has diameter: 9
NetworkX diameter is: 9

Graph out-dblp_inproceedings has diameter: 66
NetworkX diameter is: 66

Graph out-dblp_mastersthesis has diameter: 1
NetworkX diameter is: 1

Graph out-dblp_phdthesis has diameter: 2
NetworkX diameter is: 2

Graph out-dblp_proceedings has diameter: 68
NetworkX diameter is: 68
```

1.7 Grafo Unione

Avendo la lista di DataFrame, la creazione del grafo unione si riduce ai seguenti passaggi: 1. Concatenazione dei DataFrame (eliminando i duplicati) 2. Creazione del Grafo come desctritto nel paragrafi precedenti

[12]:
```
def concatenate_DataFrame_from_list(df_list:list[pd.DataFrame])->pd.DataFrame:
    df = pd.concat(
        df_list,
        axis=0,
        ignore_index=True
    )
    df.drop_duplicates(
```

```
        subset='title',
        keep='first',
        inplace=True
    )
    return df

def build_union_graph_from_DataFrame_list(df_list:list[pd.DataFrame])->nx.Graph:
    return create_graph(
        concatenate_DataFrame_from_list(df_list)
    )
```

[13]:
```
G = build_union_graph_from_DataFrame_list(df_list)
```

A questo punto avendo il grafo nella stessa forma dei precedenti, possiamo applicare le solite funzioni per trovare le risposte alle domande.

In questo caso è stata creata una funzione che comprende la chiamata a tutte le funzioni per rispondere alle domande.

[14]:
```
def answer_to_all_main_questions(G:nx.Graph,name:str)->None:
    print(f"-------------Graph: {name}--------------\n")

    title,counter = get_publication_with_max_authors(G)
    print("The publication with most authors is:")
    print(f"{title} \nWich has {counter} authors\n")

    author, count= get_author_with_most_collaborations(G)
    print(f"{author}, with {count} collaborators.\n")

    diameter = iFub(G)
    print(f"The graph diameter is: {diameter}")
```

[15]:
```
answer_to_all_main_questions(
    G,
    "Union"
)
```

```
-------------Graph: Union--------------

The publication with most authors is:
Making Bertha Drive - An Autonomous Journey on a Historic Route.
Wich has 31 authors

Edmond Bianco, with 162 collaborators.

The graph diameter is: 92
```

1.8 Autori con più collaborazioni insieme.

Per rispondere a questa domanda dobbiamo prima creare il grafo degli autori, che sarà un grafo pesato.

Durante la creazione del grafo, un arco corrisponde ad una collaboraione, ogni volta che due autori collaborano, il peso del loro arco viene incrementato di uno.

Avendolo strutturato in questo modo, il problema si riduce a trovare l'arco con il peso maggiore. Possiamo convenientemente usare una funzione che si comporta nel seguente modo: 1. Prende tutti gli archi (con i relativi dati) dal grafo. 2. Ordina *al contrario* usando come chiave il campo `weight`. 3. Prende il primo elemento della lista ottenuta.

> L'espressione `lambda x: x[2]['weight']` potrebbe confondere. Stiamo decidendo quale siano le chiavi per ordinare e stiamo dicendo che: Ad ogni elemento della lista bisogna prendere il secondo campo (partendo da 0) che corrisponde al dizionario degli attributi (0 e 1 sono il nome dei nodi comunicanti), in questo dizionario dobbiamo considerare il valore della chiave `weight`.

[16]:
```
import itertools

def build_authors_graph_from_DataFrame_list(df_list:list[pd.DataFrame])->nx.
 ↪Graph:
    df = concatenate_DataFrame_from_list(df_list)
    df = df["author"]
    G = nx.Graph()
    for authors in df:
        authors_list = authors.split("|")
        for author_comb in itertools.combinations(authors_list,2):
            if G.has_edge(*author_comb):
                G[author_comb[0]][author_comb[1]]["weight"] += 1
            else:
                G.add_edge(*author_comb,weight = 1)
    return G

def find_most_collaborating_couple(G:nx.Graph)->tuple[str,str,dict[int]]:
    return  sorted(G.edges(data=True),key= lambda x:␣
 ↪x[2]['weight'],reverse=True)[0]

author_1, author_2, weight = find_most_collaborating_couple(
    build_authors_graph_from_DataFrame_list(df_list)
    )

print(f"The most collaborating authors are {author_1} and {author_2} with␣
 ↪{weight['weight']} collaborations togheter ")
```

```
The most collaborating authors are Richard T. Snodgrass and Christian S. Jensen
with 33 collaborations togheter
```