

DEFLANDRE David
LOBJOIS Mathéo

SOUTENANCE SAE MATH GRAPHE

SOMMAIRE

01

Introduction

02

Les requêtes

03

L'application

04

Les difficultés et les succès

05

Analyse

06

Conclusion

07

Démonstration



Les requêtes

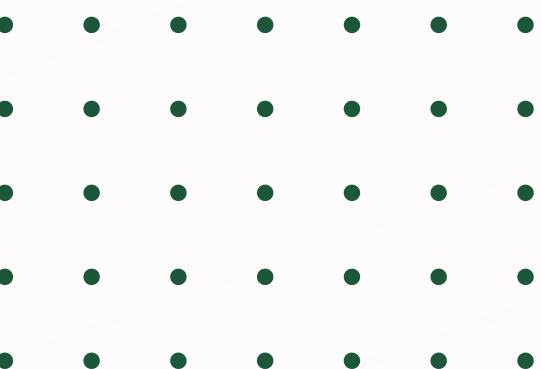
json_vers_nx

La première étape de notre travail consiste à traduire le jeu de données (au format json) représentant les collaborations en un graphe Networkx exploitable.

```
def json_vers_nx(chemin):
    """Convertit un fichier .json en graphe NetworkX."""
    with open(chemin, 'r') as f:
        films = [json.loads(line) for line in f]

    G = nx.Graph()
    for film in films:
        cast = [actor.strip('\'').replace('[[', '[').replace(']]', ']') for actor in film.get('cast', [])]
        if cast:
            G.add_edges_from((cast[0], acteur) for acteur in cast[1:])

    return G
```



Centralité

On cherche à déterminer la centralité d'un acteur
Pour cela, on veut déterminer la plus grande distance
qui le sépare d'un autre acteur dans le graphe.

```
def centralite(G, acteur):  
    # Calculer les distances les plus courtes depuis l'acteur à tous les autres nœuds  
    distances = nx.shortest_path_length(G, source=acteur)  
    # Retourner la plus grande distance trouvée  
    return max(distances.values())
```

Distances entre 2 acteurs

Tentons maintenant de déterminer la distance entre deux acteurs

```
def distance_entre_acteurs(G, acteur_source, acteur_cible):
    if acteur_source not in G.nodes or acteur_cible not in G.nodes:
        print("Au moins un des acteurs est inconnu dans le graphe.")
        return None

    queue = deque([(acteur_source, 0)]) # File contenant des tuples (acteur, distance)
    visite = set([acteur_source]) # Ensemble des acteurs déjà visités

    while queue:
        acteur_actuel, distance_actuelle = queue.popleft()

        if acteur_actuel == acteur_cible:
            return distance_actuelle

        for voisin in G.adj[acteur_actuel]:
            if voisin not in visite:
                visite.add(voisin)
                queue.append((voisin, distance_actuelle + 1))

    return -1
```

Éloignement max

Nous allons maintenant tenter de déterminer si deux acteurs peuvent être très éloignés dans Gc

```
def eloignement_max(G):
    """Retourne le couple d'acteurs ayant la plus grande distance dans le graphe."""
    max_distance = 0
    max_couple = (None, None)
    for u in G.nodes:
        for v, length in nx.single_source_shortest_path_length(G, u).items():
            if length > max_distance:
                max_distance = length
                max_couple = (u, v)
    return max_couple, max_distance
```

Application

programme principal

```
def programme_principal():
    G = demander_charger()

    root = tk.Tk()
    root.title("Analyse du Graphe Hollywoodien")

    options = [
        ("Voir le graphe", lambda: voir_graphe(G)),
        ("Voir le centre du graphe", lambda: voir_centre_graphe(G)),
        ("Voir tous les acteurs qui ont travaillé avec deux acteurs donnés", lambda: voir_acteurs_collab(G)),
        ("Voir le couple d'acteurs qui sont le plus éloignés du graphe", lambda: voir_acteurs_eloignes(G)),
        ("Voir la distance qui sépare deux acteurs", lambda: voir_distance_acteurs(G)),
        ("Voir tous les acteurs qui sont à une distance donnée d'un acteur donné", lambda: voir_acteurs_distance(G)),
        ("Voir la centralité d'un acteur", lambda: voir_centralite_acteur(G)),
        ("Arrêter le programme", root.quit)
    ]

    for text, command in options:
        button = tk.Button(root, text=text, command=command)
        button.pack(fill=tk.X, padx=20, pady=20)

    root.mainloop()

programme_principal()
```

Fonctions types

Que fait voir_centralite_acteur ?

```
def voir_centralite_acteur(G):
    acteur = demander_1_acteur(G)
    if acteur:
        result = requetes.centralite(G, acteur)
        messagebox.showinfo("Centralité d'un Acteur", result)
```

Que fait demander_2_acteurs ?

```
def demander_2_acteurs(G):
    """Demande à l'utilisateur les 2 acteurs à utiliser"""
    rep1 = simpledialog.askstring("Acteur", "Quel est le premier acteur?")
    rep2 = simpledialog.askstring("Acteur", "Quel est le deuxième acteur?")
    if rep1 in G.nodes and rep2 in G.nodes:
        return rep1, rep2
    messagebox.showerror("Erreur", "Un des deux acteurs n'est pas dans le graphe")
    return None, None
```

• • •
• • •
• • •
• • •
• • •
• • •
• • •
• • •



DIFFICULTÉS ET SUCCÈS

-

- Connaissance des outils disponibles
- Optimisation du code
- Planification

+

- Gestion du temps
- Recherches
- Implication des parties prenantes

ANALYSE

	V1	V2
<i>centralité</i>	$O(N^3)$	$O(N)$
<i>distance entre 2 acteurs</i>	$O(N^3)$	$O(n^2+m)$
<i>json_vers_nx</i>	$O(N^3)$	$O(n \cdot m^2)$
<i>éloignement max</i>	$O(VE + V^2)$	/

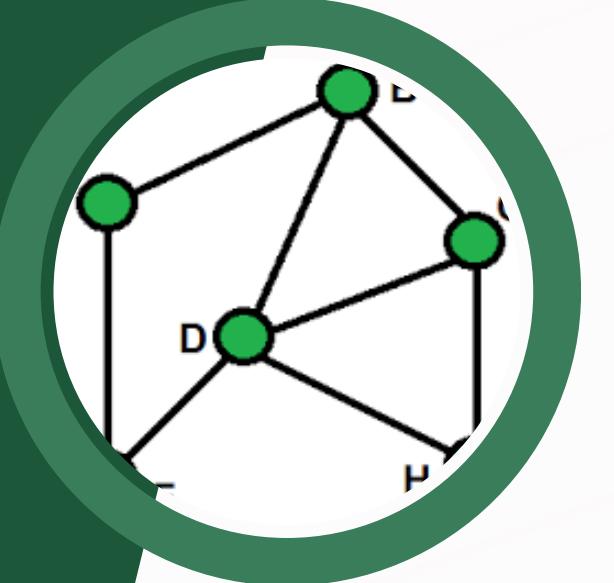
Conclusion

Optimisation
et analyse
complexité

Apprentissage
networkx,
tkinter

Gestion du
projet

Demonstration



**MERCI POUR
VOTRE
ÉCOUTE**

