Projets

Travail à faire :

- Le projet est à faire en binôme (ou trinôme)
- Choisir un des trois projets
- Écrire le programme répondant le mieux au projet demandé
- Faire une présentation orale (diaporama) de 15 minutes sur le projet : résultats, stratégie, répartition des tâches, ...
- Langage de programmation : Python, Java ou bien C/C++. Éventuellement JavaScript pour un affichage web
- Vous devrez quantifier vos résultats : statistiques, graphiques, durée calculs, qualité des résultats, critique, ...

I. Projets 1 : films + web scrapping + arbre pondéré

Une personne cinéphile possède une liste de films. On aimerait, à partir d'un programme, lui indiquer de nouveaux films dont il pourrait être intéressé en lien avec ses goûts.

Pour cela, vous devrez créer un graphe pondéré : les sommets sont les films et les arêtes sont les liens entre les films (chaque film est caractérisé par un genre, une date, une liste d'acteurs, un réalisateur, une note, ...). Ce graphe relie les films entre eux avec des arêtes pondérées : ceux qui ont une forte correspondance ont un poids proche de 1 (mêmes acteurs, même réalisateur, même genre, ...), sinon le poids est proche de 0. On pourra utiliser plusieurs stratégies de recommandation. Il est conseillé de supprimer les arcs dont le poids est inférieur à un certain seuil.
Ainsi, en parcourant ce graphe, ainsi que la liste de films donnée en entrée, on est capable de donner les films les "plus reliés". On recherche donc les sommets non découverts, fortement connectés aux sommets déjà découverts. Cependant, on souhaite aussi éviter les films trop "populaires" qui peuvent "dominer" ce graphe. Parmi les stratégies envisageables, on pourra utiliser un score des poids voisins, une densité de sommets intéressants, une clusterisation des sommets ...

Pour créer le graphe, on propose d'utiliser le web scraping pour récupérer sur le web le plus de films possible : en Python, le package Cinemagoer est approprié. On pourra stocker le graphe dans une base de données simple, ou bien dans un CSV.

exempleCinemagoer.py — Python

```python
from imdb import Cinemagoer # pip install cinemagoer

ia = Cinemagoer()

movie = ia.get_movie('0133093') # matrix

movieID = int(movie.movieID)
print("-- movieID  :",movieID)

kind =  movie.get('kind')
print("-- kind :",kind)

title = movie.get('title')
print("-- title : ", title)
```

On peut aussi envisager une page web pour afficher le graphe avec comme sommets les affiches des films.

La liste de films en entrée est de la forme :

listeFilms.txt — fichier texte

```
STAR WARS EPISODE III
STAR WARS EPISODE II
STAR WARS EPISODE I
BLADE RUNNER
ALIEN COVENANT
I ROBOT
INDIANA JONES ARCHE PERDUE
```

II. Projets 2 : OpenStreetMap + optimisation trafic routier

Il est possible de récupérer sur le web le réseau routier de toute l'Europe : c'est le projet OpenStreetMap. Les maps du monde entier sont libres, pour les télécharger : https ://download.geofabrik.de/europe.html.

On souhaite récupérer le réseau routier d'une petite ville (Chalon par exemple), et utiliser ce réseau (un graphe contenant des sommets et des arêtes) pour simuler le déplacement d'un certain nombre de véhicules : les véhicules ont un point de départ et un point d'arrivée aléatoires. Les véhicules doivent utiliser les routes définies par OpenStreetMap. Évidemment, les voitures ne peuvent pas se "rentrer dedans". On s'attend à une grande quantité de véhicules.

En Python, on pourra utiliser le package osmium :

exempleOpenStreetMap.py — Python

```python
import osmium # pip install osmium

fichierPBF="monaco.osm.pbf"
# https://download.geofabrik.de/europe.html

class RoadGraphHandler(osmium.SimpleHandler):
    def __init__(self):
        super().__init__()
        self.nodes = {}   # {node_id: (lat, lon)}
        self.edges = []   # [(node1, node2, attrs)]

    def node(self, n):
        self.nodes[n.id] = (n.location.lat, n.location.lon)

    def way(self, w):
        if "highway" in w.tags:  # garder seulement les routes
            nodes = list(w.nodes)
            for i in range(len(nodes)-1):
                n1, n2 = nodes[i].ref, nodes[i+1].ref
                attrs = {
                    "highway": w.tags.get("highway"),
                    "name": w.tags.get("name"),
                    "oneway": w.tags.get("oneway"),
                }
                self.edges.append((n1, n2, attrs))
                if attrs["oneway"] != "yes":
                    # route bidirectionnelle
                    self.edges.append((n2, n1, attrs))

handler = RoadGraphHandler()
```

continued on next page . . .

continued from last page . . .

```
handler.apply_file(fichierPBF, locations=True)

print("Nb de sommets:", len(handler.nodes))
print("Nb darêtes:", len(handler.edges))
```

Dans un deuxième temps, on souhaite ajouter des feux tricolores (feux rouges et feux verts uniquement) qui imposent l'arrêt des véhicules aux intersections. On se posera alors la question, pour chaque feu, quelle est la meilleure temporisation (feu rouge, feu vert) pour optimiser au mieux le flux des véhicules ? On analysera en particulier le temps moyen et la vitesse moyenne des véhicules.

On pourra utiliser une page web (voir projet OpenStreetMap) pour afficher cette simulation.

III. Projets 3 : OpenStreetMap + voyageur de commerce + algorithme des fourmis

Il est possible de récupérer sur le web le réseau routier de toute l'Europe : c'est le projet OpenStreetMap [1].
On souhaite utiliser le réseau routier d'un département (la Saône-et-Loire par exemple).

Dans un premier temps, on prendra une douzaine de points et on définira un itinéraire optimal pour relier chacun de ces points (Dijkstra ou A*). Ensuite, on utilisera l'algorithme des fourmis [2] pour optimiser les trajets entre tous ces sommets.
Dans un second temps, on tiendra compte du fait que le trajet doit être effectué selon les contraintes suivantes : le point de départ est défini, le conducteur doit partir et finir son trajet par ce point. Il doit respecter des limitations de vitesse sur les routes. La durée d'une journée est de 7 heures maximum. S'il ne peut pas effectuer le trajet sur une journée, il recommence le lendemain (mais avec le même point de départ). Le but est d'effectuer les trajets en le moins de temps possible (on pourra aussi calculer le trajet le plus court possible).

On pourra aussi considérer non plus un conducteur, mais plusieurs conducteurs (compagnie de transport). Combien de conducteurs minimum sont nécessaires pour effectuer tous ces arrêts ?
On pourra aussi considérer le cout (salaire conducteur, prix kilométrique, usure véhicule, . . .)

1. voir exempleOpenStreetMap.py ci dessus
2. voir Wikipédia