Université Sorbonne Paris Nord

SAE 3.01 – Base de données de films

Algorithme de rapprochement de films

Team DraCorporation

L'algorithme de rapprochement des films, c'est avérer difficile en conception et a demandé beaucoup de réflexions avant même son développement. Nous sommes partis voir plusieurs professeurs afin de savoir comment créer cet algorithme.

Je tiens à remercier M.Hebert et M.Boutinon pour leur conseil. C'est grâce à eux si on a pu développer et avoir une bonne conception du dernier algorithme.

Pour créer cet algorithme, j'ai voulu utiliser python, car je suis plus à l'aise avec ce langage et c'est un langage puissant avec ces différents modules qui existent.

Pour ce dernier algorithme, j'ai donc créé 2 programmes : le premier programme se nomme CreationGraphe.py et le second programme se nomme Algorithme_Rapprochement_Des_Films.py.
Commençons avec le premier programme « CreationGraphe.py » qui permet de créer un graphe afin de trouver le plus court chemin entre un acteur/film et un autre acteur/film.

Ce premier programme utilise les modules suivants :

Modules	**Utilités**
Psycopg2	Permets de nous connecter à la base de données et d'avoir des interactions avec elle.
json	Permets de lire/écrire des fichiers efficacement et rapidement.
Time	Permets de faire des calculs de temps d'exécution d'un programme.

Dans un premier temps le programme lit le fichier credentials.txt qui contient toutes les informations de connexion à la base de données. Puis il se connecte à la base de données.

Il va ensuite créer la fonction "createGRAPHE()" qui va nous permettre de créer le graphe. Cette fonction va utiliser 3 tables différentes de la base de données :

- "titlebasics" pour récupérer tous les id (tconst) des films,
- "namebasics" pour récupérer tous les id (nconst) des acteurs/actrice,
- "titleprincipals" pour associer les id (nconst) des acteurs et les id (tconst) des films et pour associer les id (tconst) des films et les id (nconst) des acteurs.

La fonction va dans un premier temps créer un dictionnaire et ajouter en tant que clef tous les id (tconst) des films, puis il ajoutera en tant que clef encore une fois tous les id (nconst) des acteurs.
Une fois cela termine, il va s'occuper de remplir les valeurs du dictionnaire avec la 3ème requête. Il va dans un premier temps aller dans une clef et vérifier s'il correspond à un film ou à un acteur une fois ceci fait, il ajoutera en conséquence les id correspondant à la clef du dictionnaire.

Ensuite, le programme va créer une fonction "createJSON()" qui permet de créer un fichier json.

Une fois, ceci fait, on fait appel aux fonctions, dans un premier temps "createGRAPHE()" va-nous créer le graphe et va nous retourner le graphe, puis on utilisera "createJSON()" pour créer un fichier json qui stockera le graphe.

Parlons maintenant du second programme Algorithme_Rapprochement_Des_Films.py, celui-ci utilise les modules suivants :

Modules	**Utilités**
heapq	Permets d'organiser les éléments d'une liste en utilisant une structure de données appelée tas binaire, qui gère les éléments en fonction de la priorité.
json	Permets de lire/écrire des fichiers efficacement et rapidement

Le programme va créer deux fonctions le premier qui est "ouvertureJSON()" et la seconde "find_shortest_path()".

La première fonction permet de lire un fichier json, plus précisément celui qui contient notre graphe, le temps de lecture est lent, car le fichier fait 1,5go, il faut compter environ 1 minute pour lire le fichier.

La seconde fonction "find_shortest_path()" utilise donc l'algorithme BFS (Breadth-first search) (en français "algorithme de parcours en largeur") pour parcourir les nœuds du graphe. Celui-ci permet de trouver le chemin le plus court dans un graphe et c'est celle qui s'avère le plus efficace, une fois combiner avec le module "heapq" le programme gagne en vitesse et trouve le chemin rapidement. En effet sans "heapq" le programme ne gère pas correctement les files de priorité pour stocker les nœuds, "heapq" permet de structurer les données et fais gagner du temps.

La fonction enregistre la distance entre le nœud actuel et le précédent nœud. Lorsque le nœud d'arrivée est atteint, la fonction utilise ces enregistrements pour construire le plus court chemin entre les deux nœuds et il remonte les précédents nœuds pour trouver le plus court chemin.