Rapport projet Web-API en Python

# Plan

Intro

J’ai choisi le premier projet car c’était pour moi, une bonne occasion de pouvoir apprendre à manipuler un Framework web en python, et j’ai trouvé Bottle particulièrement simple et léger, contrairement à Django (dont je connais que brièvement).

Je suis aussi satisfait de savoir maintenant parser un fichier xml, de plusieurs giga-octects.

Mais le plus intéressant pour moi fut de réaliser l’algorithme de calcule de distance entre plusieurs auteurs que j’ai implémenté moi même sans aucune bibliothèque extérieur, qui marche sur le sample de données que j’ai réalisé, mais peine à terminer sur un sample plus important…

Je vais donc via ce rapport, vous expliquer les parties les plus importantes de des fichiers de codes :

* App.py
* xmlParser.py
* Tree.py
* Tree2.py

# App.py

Ce fichier est le fichier principal, utilisé par Bottle pour gérer les requêtes des clients.

Dès le lancement du serveur web, il va parser le document xml et l’ajouter dans une liste python afin d’avoir une représentation en mémoire de ce fichier (donc en ram), et ainsi éviter les lectures beaucoup plus lentes sur le disque dur lorsqu’on voudra rechercher des auteurs par exemple.

# xmlParser.py

Le fichier xmlParser possède 2 méthodes lui permettant d’enregistrer sur le disque dur en ode binaire, une liste d’auteurs en fonction de la date de publication de leur article, permettant ainsi par la suite, de charger directement cette liste, et évitant d’avoir à reparser tout ce fichier xml, ce qui est beaucoup plus rapide.

Lorsque que je rentre dans une balise xml (name), il y a 2 cas de figure possibles:

1. Balise ouvrante :
   1. De type publicationBalise :

Si le nom de la balise appartient à l’ensemble : {"article", "inproceedings", "proceedings", "book", "incollection", "phdthesis", "masterthesis"} (self.publications\_index) , alors je mets le nom de cette variable dans self.publicationBalise pour pouvoir détecter par la suite, la fermeture de cette balise principale.

* 1. De type fieldBalise :

Je fais la même chose pour les balises secondaires (qui sont les attibuts des balises principales)

1. Balise fermante
   1. De type publicationBalise :

Je sauvegarde la structure sur le disque dur puis je réinitialise toutes les variables, pour pouvoir recommencer à parser une autre balise « publicationBalise »

* 1. De type fieldBalise :

Je reinitialise uniquement la variable « fieldBalise »

# Tree.py

Création d’un arbre, permettant ensuite de calculer la distance entre deux auteurs, et donc de calculer la distance entre deux nœuds d’un arbre.

Contraintes :

* Il ne doit pas y avoir de boucle dans l’arbre, sinon on bouclerait indéfiniment.
* Optimiser la création et la recherche dans l’arbre.

L’idée est donc de créer une classe Tree, qui, lorsqu’on lui donne un auteur en argument, crée un arbre avec tous les auteurs qui ont travaillé directement avec cet auteur, mais crée aussi des arbres, de manière récursive, pour chacun de ses co-auteurs.

Par exemple, create\_tree() crée un objet « Auteur » pour chaque co-auteur, puis se rappelle elle-même de manière récursive sur cet objet, et ajoute 1 à la profondeur. L’arbre se crée donc comme un parcours en profondeur.

Le moyen utilisé pour éviter les boucles infinies, ainsi que l’empêchement de création d’un arbre trop grand, est l’utilisation d’une liste d’exclusion (exclude\_list), où l’on ajoute chaque nouvel auteur qui a déjà été dans la liste des co-auteurs (coautors) pour que lors de l’appel récursif de create\_tree(), l’arbre de l’auteur courant n’ai pas déjà été réalisé.

Une fois l’arbre réalisé, il suffit juste de faire appel à la fonction get\_depth(), de lui transmettre les deux nom d’auteur en argument, et après avoir parcouru l’arbre, elle renverra la distance entre de nœuds de celui ci, qui correspondra à la distance entre deux auteurs, ou zéro sinon.

Conclusion