UPMC
SORBONNE UNIVERSITÉS

Rapport projet Web-API en Python

Etudiant Alexis GALLEPE

Master 1 INFORMATIQUE RES

Année universitaire 2015 - 2016

Introduction

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

Je suis tout aussi satisfait d’être en mesure, maintenant, de parser un fichier XML de plusieurs giga-octets.

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

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

- App.py
- xmlParser.py
- Tree.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.

Ce fichier comporte donc toutes les routes de notre application, où chaque réponse renvoyée par le serveur sera sous format JSON, même les erreurs.

xmlParser.py

Le fichier xmlParser possède 2 méthodes lui permettant d'enregistrer sur le disque dur en mode binaire, une liste d'auteurs, en fonction de la date de publication de leur article, admettant ainsi par la suite, de charger directement cette liste, et évitant d'avoir à re-parser 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** :

a. **De type publicationBalise** :

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

b. **De type fieldBalise** :

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

2) Balise fermante

a. **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 »

b. **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 deux nœuds, qui correspondra à la distance entre deux auteurs, ou zéro sinon.

Conclusion

Je pense avoir répondu aux attentes de ce sujet, j'y ai amélioré mes connaissances en python, et surtout en algorithmique, grâce au dernier problème de distance entre auteur, que j'ai trouvé particulièrement intéressant, même s'il n'a pas l'air très efficace sur de trop grands arbres. (Peut-être est-ce la faute à python? Je serais intéressé par vos commentaires sur la façon dont j'ai utilisé mon algorithme, comparé au votre par exemple).