IFT 2015
TP 1
22 janvier 2014
(À remettre avant le 9 février minuit)

L'objectif de ce TP est à la fois de vous familiariser avec Python et de vous donner une première intuition de l'importance des structures de données. Pour ce faire, vous allez implanter 3 classes permettant de stocker des mots, et comparer l'efficacité de 3 méthodes, soient l'insertion, la suppression et la recherche, lorsqu'implantées dans les 3 structures suivantes :

1. Une classe `Liste` qui utilisera une instance de la classe `list` de Python ;
2. Une classe `ListeTriee` qui utilisera une instance de la classe `list`, mais la maintiendra triée (en ordre lexicographique croissant) et utilisera une recherche binaire (que vous devez implanter) ;
3. Une classe `Dictionnaire` qui utilisera une instance de la classe `dict` de Python (hashtable).

Les coquilles de ces 3 classes vous sont fournies. La fonction `__str__` devra retourner une liste de Tuples (clé, compte) des mots contenus dans l'objet, sous forme de `str`. Vous allez aussi créer une classe `Mot`. Un `mot` contiendra une `str cle`, ainsi qu'un `int compte` pour tenir le compte du nombre de fois que ce mot est présent dans la structure. Vous devez aussi implanter dans la classe `Mot` les méthodes `incrementer`, `decrementer`, `get_compte` et `get_cle`. Le constructeur de votre classe `Mot`, lorsqu'il ne reçoit pas de clé en paramètre, devra en générer une aléatoirement, par exemple en pigeant aléatoirement un nombre aléatoirement choisi de lettres parmi une `str` comme `abcdefghijklmnopqrstuvwxyz`. Vous devez aussi *overrider* les méthodes `__eq__` et `__lt__` pour qu'elles ne comparent leur argument qu'avec la `cle`. Une coquille de votre classe `Mot` est aussi fournie.

Alors que vos classes `Liste` et `ListeTriee` stockeront des `Mots` dans une instance de la classe `list` de Python, votre classe `Dictionnaire` les stockera dans une instance de `dict`, qui consiste en une table de hachage. De plus, alors que les méthodes `inserer`, `supprimer` et `trouver` de votre classe `Liste` pourront utiliser les méthodes fournies par la classe `list` (et similairement pour votre classe `Dictionnaire`), votre classe `ListeTriee` devra implanter la recherche binaire. On devrait donc s'attendre à ce que certaines méthodes de votre classe `ListeTriee` soient plus efficaces que celles de votre classe `Liste`, et que certaines méthodes de votre classe `Dictionnaire` soient encore plus efficaces. C'est ce que vous allez tester empiriquement.

Barème

(1 point)

Vos fichiers doivent s'exécuter sous **Python 3**, sans message d'erreur.

(3 points)

Une fonction de validation (*unit testing*) se trouve à la fin de chacun des modules de vos classes. Elle teste la validité de vos implémentations à l'aide du fichier `test1.txt` et des deux autres fichiers `.txt` fournis. Lorsque vous roulez l'une de vos classes, par exemple `python3 Liste.py`, *"Auncun bug détecté."* devrait s'afficher. Après la remise, vos scripts seront testés sur un autre fichier texte, et possiblement avec une fonction de validation différente qui testera votre code plus en profondeur.

(6 points)

Créez une instance de chacune de vos 3 classes, en utilisant la coquille `TP1.py`. Pour chaque instance, générez différents nombres de mots (e.g. des puissances de 10) avec `Mot()` (c'est-à-dire sans argument, pour que le constructeur le génère aléatoirement), et insérez-les dans votre instance. Afin de générer les mêmes mots pour les 3 classes, n'oubliez pas de *re-seeder* votre générateur (avec la même racine) pour chacune des 3 instances (e.g. `random.seed(1234)`). Vous devez calculer le temps que prendront ces insertions avec chacune des 3 structures, à l'aide de `time.time()` (n'oubliez pas d'importer les module `time` et `random`).

De la même façon, calculez le temps que prendront des suppressions et des recherches dans chacune des 3 structures. Produisez des courbes d'efficacité des 3 méthodes dans les 3 structures (1 graphique par méthode, 3 courbes par graphique pour les 3 structures). L'axe du nombre d'opérations devrait être en `log`. Qu'en concluez-vous ?

3 points seront alloués pour les 3 graphiques et de courtes conclusions, et 3 points pour la qualité du code. La qualité du code comprend l'efficacité de votre code. Il est important que vos graphiques montrent bien les temps d'exécution, afin de bien mesurer l'efficacité de votre code. Il faut éviter de faire consécutivement plusieurs fois une même opération (de recherche, par exemple).

Vous devez remettre vos fichiers `.py` (exécutables sous Python 3), ainsi qu'un fichier `.pdf` décrivant/expliquant brièvement les résultats/conclusions de votre analyse. Remettez-les dans Studium sous forme d'un dossier compressé.

Addenda

Veuillez noter que les méthodes `inserer` et `supprimer` de vos classes `Liste`, `ListeTriee` et `Dictionnaire` doivent incrémenter ou décrémenter le compte d'un `Mot` s'il est présent (et le supprimer si son compte tombe à 0). De plus, ces deux méthodes, ainsi que la méthode `trouver`, doivent retourner `True` si le `Mot` est présent, ou `False` sinon. Vous devez aussi implanter une méthode `get_mot` qui retourne le `Mot` s'il est présent ou `None` sinon.