Fiche 5.1 : complexité des structures de données

BUT Informatique
IUT de Vélizy

Dans ce TP, vous allez procéder aux mêmes **mesures de performance** et aux mêmes **affichages graphiques** que dans le TP précédent. Mais cette fois-ci, votre travail portera sur l'étude des deux principales structures de données natives de python : `list` et `dict`.
En effet, nous allons nous poser les questions suivantes (et y répondre empiriquement) :

- le **temps de construction** d'une liste et d'un dictionnaire sont-ils les mêmes ?
- l'**accès** à un enregistrement dans une liste ou un dictionnaire sont-ils aussi rapides ? Dépendent-ils de la taille du dictionnaire ?
- la **recherche** dans une liste ou un dictionnaire, via la fonction native `in`, offrent-ils les mêmes performances ?

Objectif : savoir choisir avec discernement les structures que vous utilisez dans vos programmes.
Vous travaillerez dans un fichier de votre choix, fonctionnant sur le même principe que dans le TP précédent. Inutile d'écrire des fonctions, nous allons tout écrire dans un main.

Préparatifs

☐ 0. **Intervalle** Définissez des constantes `STEP`, `NBSTEPS` et `REP` indiquant respectivement le pas de construction des structures, le nombre d'étapes, et le nombre de répétitions. Exemple, si `STEP=100`, `NBSTEPS=200` et `REP=50`, cela signifie qu'on va manipuler des structures de taille 100 à 20000 par pas de 100, et qu'on répètera les tests 50 fois afin de stabiliser les mesures.

☐ 1. **Boucle principale** Construisez une liste `lx` qui contienne toutes les tailles de structure que nous allons créer (en utilisant `STEP` et `NBSTEPS`, donc). Lancez ensuite une grande boucle `for` où `n` parcourt cette liste `lx`. Ce sera notre boucle principale.

Construction d'une structure

☐ 2. **Construction de liste** Vous allez mesurer le temps de construction d'une liste `l` de taille `n`, contenant les entiers de `0` à `n-1`, dans l'ordre. Récupérez les temps de construction dans une liste `perfConstList` que vous utiliserez, comme dans le TP précédent, pour afficher un joli graphique.

☐ 3. **Construction de dictionnaire** De la même manière, on construit un dictionnaire contenant `n` enregistrements de la forme `i:i` avec `i` compris entre `0` et `n-1`. On mesure le temps de construction et on le stocke dans une liste `perfConstDict`, puis on crée le graphique associé.

☐ 4. **Les tests** Testez votre programme avec `STEP=200`, `NBSTEPS=200`. La constante `REP` n'et pas utilisée pour le moment.

☐ 5. **Conclusions** Qu'en concluez-vous ?

Accès à une structure

☐ 6. **Accès aux listes** Dans la grande boucle précédente, nous allons ajouter, derrière les constructions, les mesure d'accès aux structures de taille variable que vous avez précédement créées. Ecrivez une boucle qui effectue `REP` tours, et, dans cette boucle, écrit simplement un 0 dans la case d'indice `len(l)//2` de la liste précédement créée. Vous calculerez le temps moyen mesuré pour cet accès et l'ajouterez dans une liste `perfAccList`.

☐ 7. **Accès aux dictionnaires** Même chose avec les dictionnaires : vous écrirez un 0 dans l'enregistrement d'indice `len(d)//2` du dictionnaire et stockerez le temps moyen pour cet accès dans une liste `perfAccDict`.

☐ 8. **Les tests** Testez votre programme global, en fixant `REP=500`, tracez les graphiques.

☐ 9. **Conclusions** Que concluez-vous ?

Recherche dans une structure

☐ 10. **Recherche dans une liste** Toujours dans la boucle principale, vous allez mesurer le temps requis pour `REP` recherches dans la liste créée précédement. Vous rechercherez un nombre qui n'y figure pas : -1. Pour utiliser la fonction native `in` de python, vous avez juste à écrire `-1 in l`. Stockez les performances moyennes dans `perfRechList`.

☐ 11. **Recherche dans un dictionnaire** Même chose dans les dictionnaires. Vous stockerez les performances dans une liste `perfRechDict`.

☐ 12. **Les tests** Lancez vos tests et graphiques.

☐ 13. **Conclusions** Qu'en concluez-vous ?