DS 2

Informatique pour tous, deuxième année

Julien Reichert

Exercice 0 (question de cours) : Écrire un des algorithmes de tri parmi le tri fusion et le tri rapide, au choix dans sa version en place ou non en place. Prouver sa terminaison ; calculer sa complexité dans le pire des cas par une analyse rigoureuse. Prouver aussi la correction de l'algorithme. En plus de cela, l'algorithme doit avoir un argument booléen précisant si le tri doit se faire dans l'ordre croissant ou non. La fonction `reversed` et la méthode `reverse` sont interdites, de même que la méthode `sort` et toutes les autres astuces du genre, bien évidemment.

Exercice 1 (pas sur les tris) : Calculer à la main le nombre de « multi-ensembles » formés de quatre nombres de 1 à 10. Il s'agit en pratique du nombre de 4-listes pouvant avoir des répétitions mais où l'ordre n'est pas important, ou encore du nombre de 4-listes croissantes. Écrire un programme, récursif ou non, qui les engendre tous (retournant une liste de listes croissantes de préférence, mais on n'impose pas qu'elles soient triées).

Vu que le calcul est une question de mathématiques, il ne donne pas beaucoup de points...

Exercice 2 : Écrire en Python une fonction implémentant le « tri Lyes » : on crée une copie de la liste de la bonne taille et on place chaque élément à une position qui se déduit du nombre d'éléments inférieurs. Attention à bien gérer les cas d'égalité. Donner la complexité de la fonction écrite.

Le tri n'est bien entendu pas en place, mais il est intéressant de constater qu'on peut le rendre stable.

Exercice 3 : Écrire en Python une fonction implémentant le « tri Dubos » : on crée une liste supplémentaire de taille exponentielle en la taille de la liste, on place le premier élément de la liste au milieu de la liste supplémentaire et les suivants à une place libre déterminée par dichotomie. Une fois tous les éléments placés, on parcourt la liste supplémentaire et on récupère tous les éléments de la liste d'origine. Inutile de donner la complexité...

Remarque : Il s'agit simplement du tri par ABR, aussi efficace que les tris qui suivent le paradigme « diviser pour régner » mais en implémentant de manière totalement non optimale la structure d'ABR. Le besoin de connaître la structure d'ABR et l'absence d'avantages sur les tris fusion et rapide fait que ce tri reste assez méconnu.

Exercice 4 : Soient les trois algorithmes suivants, agissant sur une liste. Déterminer pourquoi les deux premiers ne sont pas des tris corrects, prouver que le troisième en est un (en particulier qu'il termine) et calculer sa complexité.

On suppose déjà écrites une fonction `engendrer_transpositions(n)` qui retourne en temps quadratique en `n` la liste des couples `(i,j)` pour $0 \leq$ `i` $<$ `j` $<$ `n`, une fonction `melange(l)` qui mélange en temps linéaire en la taille de `l` son argument de façon pseudo-aléatoire et une fonction `croissante(l)` qui détermine en temps linéaire en la taille de `l` si son argument est une liste croissante.

```
def petit_tri_ah_non(liste):
    transpo = engendrer_transpositions(len(liste))
    melange(transpo)
    while not (croissante(liste)):
        (i,j) = transpo.pop()
        liste[i], liste[j] = liste[j], liste[i]
```

```
def grand_tri_ah_non(liste):
    transpo = engendrer_transpositions(len(liste))
    melange(transpo)
    while not (croissante(liste)):
        (i,j) = transpo.pop()
        if liste[i] > liste[j]:
            liste[i], liste[j] = liste[j], liste[i]

def tri_castin(liste):
    transpo = engendrer_transpositions(len(liste))
    melange(transpo)
    k = 0
    while not (croissante(liste)):
        (i,j) = transpo[k]
        if liste[i] > liste[j]:
            liste[i], liste[j] = liste[j], liste[i]
            k = 0
        else:
            k += 1
```

Exercice 5 : On considère la base de données suivante, utilisée par une auto-école afin de gérer les sessions d'exercices au code. Les tables sont les suivantes, avec des attributs intuitifs :

- `CLIENTS`, avec les attributs `Id` (entier, clé primaire), `NomPrenom` (chaîne de caractères) et d'autres informations qui ne nous intéressent pas ici ;
- `SEANCES`, avec les attributs `Id_seance` (entier, clé primaire), `Date` (format spécifique ordonné), `Id_client` (entier, clé extérieure référant au client), `NbFautes` (entier) ;
- `REPONSES`, avec les attributs `Id_seance` (entier, clé extérieure référant à la séance), `Id_client` (entier, clé extérieure référant au client), `Question` (entier), `Reponses` (chaîne de caractères) ;
- `SOLUTIONS`, avec les attributs `Id_seance` (entier, clé extérieure référant à la séance), `Question` (entier), `Reponses` (chaîne de caractères) ;

Question 1 : Quelle clé primaire peut-on envisager pour les tables `REPONSES` et `SOLUTIONS` ?

Question 2 : Au vu de la structure, on peut considérer qu'une table n'était pas nécessaire. Comment procéder pour s'en passer ?

Question 3 : Écrire des requêtes permettant de récupérer les informations suivantes :

- Le nombre de clients enregistrés.
- L'ensemble des dates où une séance a eu lieu.
- Le nombre moyen de fautes d'un client précisé (on considère que son identifiant est disponible dans la valeur `n`).
- Le nombre minimal de fautes sur l'ensemble des clients et des séances.
- Le nombre maximal de questions ayant la même solution lors d'une séance précisée (on considère que son identifiant est disponible dans la valeur `s`).
- Le nombre de fautes qu'un client précisé (on considère que son identifiant est disponible dans la valeur `n`) a faites lors d'une séance précisée (de même, l'identifiant sera `s`). Pour cette dernière requête, on s'interdira d'utiliser la table `SEANCES` (l'idée est de faire une requête d'insertion à partir de données élémentaires).

Question 4 : Un client est considéré comme prêt à passer l'examen du code s'il a toujours fait moins de cinq fautes lors de ses trois précédentes séances. Écrire une requête qui détermine si c'est le cas d'un client précisé.

Question 5 : L'auto-école a décidé d'une offre de lancement promotionnelle pour fêter les deux mois de son activité : au client qui a fait le moins de fautes au total et dont l'assiduité est parfaite, dix heures de conduite sont offertes. Écrire une requête qui a permis de le trouver. On pourra supposer qu'il y a existence et unicité.