Correction du DS 2

Informatique pour tous, deuxième année

Julien Reichert

Exercice 1

Question de cours, pas de correction.

Exercice 2

Une façon simple est de localiser à chaque étape le minimum parmi les valeurs non encore définitivement placées et de faire deux renversements : un qui place le minimum en question en fin de liste, et un qui le place au bon endroit. On a donc un algorithme de tri par insertion avec les opérations élémentaires restreintes.

```
def renversement(l, debut):
    if debut == 0:
        l[:] = l[::-1] # ne pas oublier [:] sinon l ne mute pas !
    else:
        l[debut:] = l[:debut-1:-1] # attention, bug si debut vaut 0

def minimum(l, debut):
    rep = debut
    for i in range(debut+1, len(l)):
        if l[i] < l[rep]:
            rep = i
    return rep

def tri_crepes(l):
    for i in range(len(l)):
        ind = minimum(l, i)
        if ind != i: # gain de temps
            renversement(l, ind)
            renversement(l, i)
```

La complexité se voit facilement (mais on a encore un nombre quadratique de comparaisons et d'affectations d'un élément), et la preuve de correction découle du tri par insertion.

Exercice 3

```
import math

def tri(l, debut=0, fin=None):
    if fin == None:
        fin = len(l)-1
    if debut >= fin-1:
        if l[debut] > l[fin]:
            l[debut], l[fin] = l[fin], l[debut]
        return
    largeur = fin-debut+1
    tri(l, debut, debut+math.ceil((2*largeur)/3-1))
    tri(l, fin-math.ceil((2*largeur)/3-1), fin)
    tri(l, debut, debut+math.ceil((2*largeur)/3-1))
```

Pour la terminaison, on se sert du variant `fin - debut`, qui décroît strictement sur chaque appel récursif imbriqué, car la « largeur » est réduite d'un tiers, avec arrêt quand elle passe à 2. Pour la correction, on peut montrer par récurrence que `tri(l, debut, fin)` trie la zone de `l` entre `debut` et `fin` inclus, et pour établir l'hérédité on prouve que les appels récursifs ont les effets successifs suivants :

- Remarque préliminaire : on peut considérer l'intervalle (de taille fixée à $3k+r$ avec $0 \leq r < 3$) comme coupé en trois zones, de tailles respectives k, $k+r$ et k et notées A, B et C.
- Premier appel : Considérons un élément parmi le tiers des éléments les plus grands de l'intervalle. Soit il déjà dans la zone C, soit il a été éjecté de la zone A, car il y a au plus $k-1$ éléments qui lui sont supérieurs et donc pas assez pour occuper toute la zone B.
- Deuxième appel : Considérons toujours le tiers des éléments les plus grands. Ils sont tous dans les zones B et C, qui subissent un tri par hypothèse de récurrence. Alors désormais, la zone C contient le tiers des éléments les plus grands dans l'ordre croissant.
- Troisième appel : Les autres éléments sont triés eux aussi dans l'ordre croissant, CQFD.

La complexité se calcule à l'aide de la formule $c_n = 3c_{\frac{2n}{3}} + \mathcal{O}(n)$, ce qui se résout en $c_n = \mathcal{O}(n^{\log_{\frac{3}{2}} 3})$, ce qui n'est vraiment pas efficace (d'où le nom anglais de *Stoogesort*, en référence à un très ancien film).

Exercice 4

Le premier algorithme teste toutes les transpositions dans un ordre arbitraire et vérifie après chaque transposition si la liste est triée. Il est facile de trouver un contre-exemple minimal : une liste de trois éléments qui n'est pas triée après chacune des trois transpositions quand l'ordre est malheureux. On note que dans ce cas, on entre dans la boucle avec une liste de transpositions restantes vide, ce qui provoquera une erreur.

```
l = [3, 2, 1]
transpo = [(0, 2), (0, 1), (1, 2)] # après mélange (et le pop prend l'élément de droite)
l <- [3, 1, 2] puis l <- [1, 3, 2] puis l <- [2, 3, 1]
```

En pratique, c'était normal qu'un contre-exemple se trouve facilement : en faisant trois transpositions, on engendre au plus quatre des six permutations existantes (en comptant celle dont on dispose au début), parfois celle qu'on cherche n'est pas parmi ces quatre-là.

Le deuxième algorithme ne fait pas les transpositions qui créent une inversion (rappel : une inversion est un couple de positions pour lesquelles les éléments sont dans le mauvais ordre). Tant pis pour lui, le résultat est le même, car avec ce contre-exemple on produit les mêmes étapes, sauf la dernière qui n'est pas faite, mais quoi qu'il arrive la liste n'est toujours pas triée.

Le troisième algorithme est un tri valide mais encore plus stupide que le tri à bulles. Il repose sur l'invariant de boucle suivant : après un passage dans la boucle conditionnelle (qui coûte un $\mathcal{O}(n)$ dans la foulée), soit on a effectué une transposition qui a fait disparaître au moins une inversion dans la liste, sans en créer d'autres (éventuellement quelques inversions en sont devenues des autres, mais le nombre total a strictement décru), soit le nombre `k` a augmenté. Dans le premier cas, on progresse en direction d'une liste sans inversion, ce qui est le cas si, et seulement si, la liste est triée, provoquant l'arrêt de l'algorithme. Dans le deuxième cas, on finira par parcourir l'ensemble des transpositions possibles, mais dès que les indices concernés correspondent à une inversion, on arrive dans le premier cas, de sorte qu'on ne déclenche jamais d'erreur de débordement de la liste des transpositions dans la mesure où après chaque transposition effectivement faite on remet `k` à zéro et surtout on ne supprime pas la transposition de la liste.

Ainsi, on a réussi à combiner une preuve de terminaison et de correction, et la complexité se lit ainsi : $\mathcal{O}(n^2)$ passages entre deux transpositions effectuées, $\mathcal{O}(n^2)$ transpositions éventuellement nécessaires plus la vérification que la liste est triée avant de faire n'importe quelle étape, ce qui veut dire que chacun des $\mathcal{O}(n^4)$ tours de boucle a un coût en $\mathcal{O}(n)$. On ne rêve pas : le tri est en $\mathcal{O}(n^5)$. Ceci étant, il s'appuie sur un tri qui frôle l'imbécillité (pour ne pas dire qui est en plein dedans) : tester toutes les permutations, et ce dernier algorithme serait en $\mathcal{O}(n!)$.

Exercice 5

Question 1 : Pour la table `Minitournoi_inscription`, tous les couples contenant `Id` (sauf avec `Parti`) peuvent être une clé, vu qu'on ne souhaite pas avoir deux personnes ayant le même numéro (peu pratique vu qu'on a créé l'attribut en question exprès pour la concision lors de jointures avec d'autres tables), le même nom (sinon le classement est ambigu) et le même identifiant d'utilisateur non nul (vu qu'il est de toute façon personnel). Pour la table `Minitournoi_serie`, on a naturellement la clé `(Id, Serie, Numero)` (on n'entre qu'un résultat par joueur à chaque série), mais on peut aussi ajouter la clé `(Id, Serie, Num_table, Place)` (un seul joueur peut s'asseoir à une même place à une série donnée).

Question 2 : `SELECT COUNT(*) FROM Minitournoi_inscription AS MI JOIN Minitournoi AS M ON MI.Id = M.Id WHERE Titre = "Tournoi de Villemomble #1"`

Question 3 : `SELECT SUM(Resultat) FROM Minitournoi_inscription AS MI JOIN Minitournoi_serie AS MS ON MI.Id = MS.Id AND MI.Numero = MS.Numero WHERE Nom = "Julien Reichert" AND MI.Id=5`

Question 4 : `SELECT MAX(Resultat) FROM Minitournoi AS M JOIN Minitournoi_serie AS MS ON M.Id = MS.Id WHERE Club=2`

Question 5 : `SELECT SUM(Resultat) AS Total FROM Minitournoi_serie WHERE Id=2 GROUP BY Numero ORDER BY Total DESC LIMIT 1`

Question 6 : `SELECT Id_utilisateur FROM Minitournoi_inscription AS MI JOIN Minitournoi_serie AS MS ON MI.Id = MS.Id AND MI.Numero = MS.Numero WHERE Id_utilisateur IS NOT NULL GROUP BY Id_utilisateur ORDER BY SUM(Resultat) DESC LIMIT 1` [1]

Question 7 : `SELECT Nom FROM Minitournoi_inscription AS MI JOIN Minitournoi_serie AS MS ON MI.Id = MS.Id AND MI.Numero = MS.Numero WHERE MI.Id=t AND (Parti = 0 OR Parti >= n) GROUP BY MI.Numero ORDER BY SUM(Resultat) DESC, SUM(Gagnes) DESC, SUM(Perdus)`

Question 8 : `SELECT Nom FROM Minitournoi_inscription GROUP BY Nom HAVING COUNT(*) = (SELECT MAX(Participations) FROM (SELECT COUNT(*) AS Participations FROM Minitournoi_inscription GROUP BY Nom) AS td)`

1. Une version plus rigoureuse utilisant `HAVING` est possible, mais plus longue. On peut voir la question 8 pour un exemple.