Mines-Ponts
2016

Informatique Pour Tous
Toutes filières
Questions Q1 à Q9 (Tri et base de données)

Corrigé

Question Q1.

On aura pu remarquer, mais ce n'est en rien nécessaire pour répondre à cette question, que le tri fourni est un tri par insertion.

Ici, on a $n = 5$ et le compteur `i` prend successivement les valeurs 1, 2, 3 et 4.

Le contenu de la liste `L` à l'issue des quatre itérations est :

```
[2,5,3,1,4]
[2,3,5,1,4]
[1,2,3,5,4]
[1,2,3,4,5]
```

Question Q2.

Le compteur `i` varie entre 1 et $n-1$.
Montrer que « La liste `L[0:i+1]` est triée par ordre croissant à l'issue de l'itération `i` » est un invariant de boucle revient à montrer que cette proposition est vraie pour toute valeur de `i` dans l'intervalle $[\![1 ; n-1]\!]$.
On va montrer ce résultat par une récurrence finie en supposant que l'on a $n \geq 2$ (sans quoi, la liste est déjà triée et la boucle n'est pas exécutée).

Pour `i=1`, on s'intéresse à la sous-liste `L[0:2]`, c'est-à-dire, en entrée de boucle, à la liste `[L[0],L[1]]`.
Avant la boucle `while`, le compteur `j` prend la valeur de `i`, c'est-à-dire 1, et la variable `x` prend la valeur `L[1]`.
La condition « `0 < j` » revient à « `0 < 1` » et est donc satisfaite.

La condition « `x < L[j-1]` » revient à « `L[1] < L[0]` ». On a donc deux situations possibles :

- `L[1] > L[0]`. Dans ce cas la boucle `while` n'est pas exécutée. La dernière instruction « `L[j] = x` » revient à « `L[1] = L[1]` ». Finalement, la sous-liste `L[0:2]` n'a pas été modifiée et est bien triée dans l'ordre croissant.
- `L[1] < L[0]`. Dans ce cas, on exécute une première fois la boucle `while` (les deux conditions sont satisfaites). « `L[j] = L[j-1]` » revient à « `L[1] = L[0]` » et la sous-liste `L[0:2]` correspond désormais à `[L[0],L[0]]`. Puis « `j = j-1` » donne la valeur 0 au compteur `j`. Ainsi, la boucle `while` ne sera pas exécutée une seconde fois puisque la première condition n'est plus satisfaite.
  La dernière instruction « `L[j] = x` » revient à « `L[0] = L[1]` » et la sous-liste `L[0:2]` correspond désormais à `[L[1],L[0]]` qui est bien triée dans l'ordre croissant.

D'après ce qui précède, notre raisonnement est initialisé. Plus précisément, si $n = 2$, le raisonnement est achevé et la liste `L` est bien triée. Dans ce qui suit, on suppose donc que l'on a $n \geq 3$.

Supposons désormais que la proposition soit vraie pour une valeur quelconque `k` (nous introduisons cette nouvelle notation afin de distinguer le nom du compteur `i` de sa valeur pour une meilleure compréhension de l'analyse) de `i` dans l'intervalle $[\![1; n-2]\!]$. En d'autre terme, à l'issue de l'itération `k`, la sous-liste `L[0:k+1]` est triée dans l'ordre croissant.
Détaillons alors l'exécution `k+1` de la boucle `for`. Notre objectif est de montrer qu'à l'issue de cette exécution, la sous-liste `L[0:k+2]` sera triée dans l'ordre croissant.

Le compteur `j` prend la valeur du compteur `i`, c'est-à-dire `k+1`.
La variable `x` prend la valeur `L[i]`, c'est-à-dire `L[k+1]`.
Comme $k \geq 1$, la condition « `0 < j` » est satisfaite.
La condition « `x < L[j-1]` » revient à « `L[k+1] < L[k]` ». On a donc deux situations possibles :

- `L[k+1] > L[k]`. Dans ce cas la boucle `while` n'est pas exécutée. La dernière instruction « `L[j] = x` » revient à « `L[k+1] = L[k+1]` ». Finalement, la sous-liste `L[0:k+1]` n'a pas été modifiée. Par hypothèse de récurrence, elle est triée dans l'ordre croissant. Or, on a `L[k+1] > L[k]`. On en déduit donc que la sous-liste `L[0:k+2]` est aussi triée dans l'ordre croissant.
- `L[k+1] < L[k]`. Dans ce cas, les deux conditions sont satisfaites et on exécute la boucle `while`. A chaque exécution, l'élément `L[j-1]` de la sous-liste `L[0:k+1]` est décalé d'un cran vers la droite (instruction « `L[j]=L[j-1]` »). Ce décalage a lieu tant que les deux conditions sont satisfaites.
  - Si le compteur `j` atteint la valeur 0, à la dernière exécution de la boucle `while` on a donc eu le décalage « `L[j]=L[j-1]` » avec `j=1`, c'est-à-dire « `L[1]=L[0]` ». La sous-liste `L[0:k+2]` est donc dans l'état `[L[0],L[0],L[1],L[2],…,L[k],L[k+1]]` (ATTENTION, les trois points « … » indiquent ici des éléments de la liste et n'ont rien à voir avec une quelconque syntaxe Python). L'instruction « `L[j]=x` » correspond à « `L[0]=L[k+1]` » et la sous-liste `L[0:k+2]` est donc finalement dans l'état `[L[k+1],L[0],L[1],L[2],…,L[k],L[k+1]]`. Mais la condition

« `x < L[j-1]` », c'est-à-dire « `L[k+1] < L[0]` », ayant également été satisfaite, on en déduit que la sous-liste `L[0:k+2]` est bien triée dans l'ordre croissant.

- Si le compteur `j` n'atteint pas la valeur 0, à la dernière exécution de la boucle `while` on a donc eu le décalage « `L[j]=L[j-1]` » pour une certaine valeur de `j` strictement supérieure à 1. La sous-liste `L[0:k+2]` est donc dans l'état `[L[0],L[1],…,L[j-1],L[j-1],L[j],…,L[k-1],L[k]]`.
  La dernière instruction « `L[j]=x` », correspond à « `L[j]=L[k+1]` » et conduit à la sous-liste `L[0:k+2]` suivante :
  `[L[0],L[1],…,L[j-1],L[k+1],L[j],…,L[k-1],L[k]]`.
  La condition « `x < L[j-1]` », c'est-à-dire « `L[k+1] < L[j-1]` », n'étant pas satisfaite, on a donc `L[k+1] < L[j]` et `L[k+1] > L[j-1]`.
  On en déduit ainsi que la sous-liste `L[0:k+2]` est bien triée dans l'ordre croissant.

De ce qui précède, on déduit que la proposition « La liste `L[0:i+1]` est triée par ordre croissant à l'issue de l'itération `i` » est héréditaire.

Notre raisonnement est donc achevé et on conclut que la proposition « La liste `L[0:i+1]` est triée par ordre croissant à l'issue de l'itération `i` » est vraie pour toute valeur de i dans l'intervalle $[\![1;n-1]\!]$. Nous avons donc bien affaire à un invariant de boucle.

En particulier, pour $i=n-1$, le résultat précédent nous permet d'affirmer que la liste `L[0:i+1]` est triée dans l'ordre croissant. Or, il s'agit de la liste `L` elle-même. Le code fourni trie bien la liste passée en argument dans l'ordre croissant.

Question Q3.

Dans cette question, nous évaluons classiquement la complexité d'un algorithme de tri comme le nombre de comparaisons effectuées entre éléments de la liste à trier.

Quelle que soit la longueur n ($n \geq 1$) de la liste `L`, la boucle `for` sera exécutée $n-1$ fois.
A l'entrée de la boucle `while` on a, si la première condition (« `0 < j` ») est satisfaite, une comparaison entre deux éléments de la liste (condition « `x < L[j-1]` »).

Dans le meilleur des cas, <u>pour chaque valeur possible du compteur `i`</u>, on a la condition « `x < L[j-1]` » qui n'est pas satisfaite et donc la boucle `while` qui n'est pas exécutée. On aura ainsi évalué $n-1$ fois la condition « `x < L[j-1]` » et il en découle une complexité en $\mathrm{O}(n)$.

Dans le pire des cas, la boucle `while` est exécutée `i` fois (`j` prend toutes les valeurs de `i` à 1). La condition « `x < L[j-1]` » est ainsi évaluée (et satisfaite) `i` fois également.
La complexité est alors égale à : $\sum_{i=1}^{n-1} i = \frac{(n-1)\times((n-1)+1)}{2} = \frac{n(n-1)}{2}$ et est donc en $\mathrm{O}(n^2)$.

Le tri fusion est plus efficace que le tri par insertion dans le pire des cas car sa complexité est en $O(n\ln(n))$. Il s'agit également de sa complexité moyenne et de sa complexité dans le meilleur des cas. Ainsi, pour une grande liste déjà triée, le tri par insertion sera plus rapide que le tri fusion. Mais est-ce vraiment une remarque dont on doit tenir compte ? ☺

Question Q4.

Dans cette question, on se sert bien évidemment du code fourni que l'on adapte afin que les comparaisons se fassent au niveau du deuxième élément de chaque élément de la liste principale. Dans le code ci-dessous, les lignes modifiées apparaissent en rouge ainsi que quelques commentaires ajoutés pour la lisibilité/compréhension.

```
def tri_chaine(L): # Nouveau nom fourni dans l'énoncé
    n = len(L)
    for i in range(1,n):
        j = i
        x = L[i] # Ici, x est une liste !
        while 0 < j and x[1] < L[j-1][1]:
            L[j] = L[j-1]
            j = j-1
        L[j] = x
```

Question Q5.

Aucun des trois attributs `nom`, `iso` et `annee` ne peut servir de clé primaire : un pays (donc son `nom` et son `iso`) peut apparaître plusieurs fois (comme le Brésil par exemple) et il en va de même pour une année (l'année 2010 par exemple).

En revanche, on dispose d'une information unique (champs `cas` et `deces`) pour un pays et une année donnés. On peut donc considérer comme clé primaire le couple (`nom`,`annee`) ou le couple (`iso`,`annee`).

Question Q6.

```
SELECT *
FROM palu
WHERE annee=`2010` AND deces>=`1000`
```

Question Q7.

On peut utiliser conjointement les deux tables `palu` et `demographie` avec ou sans jointure. Afin de rendre le résultat lisible, nous sélectionnons l'attribut `nom` et nous aliasons le calcul du taux d'incidence.

Sans jointure :

```
SELECT nom,deces/pop*10000 AS Taux Incidence
FROM palu,demographie
WHERE iso=pays AND annee=2011 AND periode=2011
```

Avec jointure :

```
SELECT nom,deces/pop*10000 AS Taux Incidence
FROM palu JOIN demographie
     ON iso=pays AND annee=periode
WHERE annee=2011
```

Question Q8.

Le plus grand nombre de nouveaux cas de paludisme en 2010 pourrait être obtenu par la requête :

```
SELECT MAX(cas)
FROM palu
WHERE annee=`2010`
```

De fait, le deuxième plus grand nombre de nouveaux cas de paludisme en 2010 pourrait être obtenu par la requête :

```
SELECT MAX(cas)
FROM palu
WHERE annee=`2010`
      AND cas<>(SELECT MAX(cas)
                FROM palu
                WHERE annee=`2010`)
```

Finalement, pour obtenir le nom du pays ayant eu le deuxième plus grand nombre de nouveaux cas de paludisme en 2010, on pourrait utiliser la requête :

```
SELECT nom
FROM palu
WHERE annee=`2010`
      AND cas=(SELECT MAX(cas)
               FROM palu
               WHERE annee=`2010`
                     AND cas<>(SELECT MAX(cas)
                               FROM palu
                               WHERE annee=`2010`))
```

Question Q9.

Rappelons ici qu'en terme d'algèbre relationnelle, l'expression $\sigma_{\texttt{annee=2010}}(\texttt{palu})$ signifie que l'on travaille sur la table `palu` et que l'on s'intéresse aux enregistrements où le champ `annee` prend la valeur 2010 (on effectue une sélection). Parmi ces champs, on ne retient que les attributs `nom` et `deces` : on effectue une projection (c'est le sens de $\pi_{\texttt{nom,deces}}$).

Le tri des couples sera simplement obtenu grâce à l'instruction :

```
tri_chaine(deces2010)
```