Preuve de tris

Des élèves ont à leur disposition les fonctions echange, decale et trouve (voir page suivante). Ils doivent proposer une fonction permettant de trier un tableau initialisé avec une permutation des entiers $[1, n]$.

Trois implémentations vous sont rendues et vous devez les évaluer. Pour chacune des trois implémentations :

1. Testez le code avec de petits exemples.
   Essayez par exemple avec les instances suivantes : $\{3, 4, 1, 2\}$, $\{4, 1, 3, 2\}$, ...
2. Montrez que le code proposé termine.
3. Montrez que le code proposé conduit au résultat attendu.

```python
def tri_insertion(tab):
    """Tri un tableau par insertion

    Les éléments successifs du tableau ``tab`` sont insérés un par un
    dans la partie triée du tableau jusqu'à ce que le tableau soit
    entièrement trié. Le tri se fait en place.

    :param tab: un tableau
    :return: le tableau ``tab`` trié (sans copie du tableau)
    :rtype: tableau
    """
    for i in range(1,len(tab)):
        j=i
        while j>0 and tab[j] < tab[j-1]:
            echange(tab, j, j-1)
            j=j-1
    return tab
```

```python
def tri_mystere1(tab):
    """Tri un tableau contenant une permutation des entiers (1..n)"""
    for val in range(1, len(tab)+1):
        pos_val = trouve(tab, val)
        decale(tab, pos_val, val)
    return tab
```

```python
def tri_mystere2(tab):
    """Tri un tableau d'entiers"""
    fini = False
    while not fini:
        fini = True
        for k in range(len(tab)-1):
            if tab[k] > tab[k+1]:
                echange(tab, k, k+1)
                fini = False
    return tab
```

Fonctions à disposition des élèves

```python
def echange(tab, i, j):
    """Permute les éléments d'indice i et j dans le tableau tab (en place)

    Les indices doivent être valides pour le tableau ``tab``

    :param tab: un tableau
    :param i: un indice du tableau
    :param j: un indice du tableau
    :return: None
    :rtype: NoneType
    """
    temp=tab[i]
    tab[i]=tab[j]
    tab[j]=temp

def decale(tab, pos, l):
    """Déplace un élément d'un tableau

    L'élément d'indice ``pos`` est déplacé de ``l`` positions dans le
    tableau (en direction de la fin du tableau). Si la position finale
    dépasse la fin du tableau, alors l'élément est placé en dernière
    position dans le tableau.

    :param tab: un tableau
    :param pos: un indice du tableau
    :param l: le nombre de décalages à faire
    :return: None
    :rtype: NoneType

    :Example:

    >>> decale([2, 5, 7, 8], 1, 1) # décale l'élément 5 (indice 1) de 1 position
    [2, 7, 5, 8]
    >>> decale([2, 5, 7, 8], 1, 6) # 5 n'est décalé que de 2 positions (il arrive au bout)
    [2, 7, 8, 5]
    """
    nb_deplacements = min(l, len(tab)-pos-1)
    for k in range(nb_deplacements):
        echange(tab, pos+k, pos+k+1)

def trouve(tab, elt):
    """Cherche un élément dans un tableau

    L'élément ``elt`` est recherché dans le tableau ``tab``. Lorsqu'il
    est trouvé, son indice est renvoyé. Si l'élément n'est pas trouvé,
    une erreur est générée.


    :param tab: un tableau
    :param elt: un élément du tableau
    :return: l'indice de l'élément cherché dans le tableau
    :rtype: int
    """

    for k in range(len(tab)):
        if tab[k] == elt:
            return k;
    raise "Error"
```

Preuve du tri par insertion

Preuve de terminaison

L'algorithme proposé a deux boucles imbriquées.

La boucle extérieure est une boucle *for* avec un nombre fixé d'itération (la longueur du tableau moins 1) donc elle ne pose pas de problème pour la terminaison.

La boucle intérieure est une boucle *while* potentiellement infinie. On peut toutefois facilement borner le nombre d'itération par la taille du tableau car :

— j est initialisé avec la valeur de i, avec $i \in [1, n[$ où n est la longueur du tableau à trier ;

— dans la boucle, la variable j n'est modifiée qu'à la ligne 17 qui

  — est exécutée à chaque tour de boucle ;

  — a pour effet de décrémenter de 1 la valeur de la variable j.

  Les valeurs de j forment donc une suite d'entiers strictement décroissante avec une valeur initiale strictement positive ;

— lorsque la valeur de j devient négative ou nulle, on sort de la boucle (ligne 15).

Il est donc impossible de boucler indéfiniment avec cette boucle interne.

Preuve de correction

Les instructions sont correctes

Il faut d'abord se convaincre que toutes les instructions sont correctes. Dans le code proposé, il peut y avoir deux problèmes :

1. des accès hors des bornes du tableau. Ici, nous avons des accès aux éléments d'indice j et $j-1$ à la ligne 15 ;
2. des appels à la fonction echange avec des paramètres incorrects. Ici, echange est appelé avec le tableau *tab* et les indices j et $j-1$.

Dans les deux cas, il s'agit de vérifier que les indices j et $j-1$ sont valides pour le tableau *tab* au moment des accès/appels, c'est-à-dire que $0 \leq j-1 \leq$j$< n$ où n est la longueur du tableau *tab* à trier, ou, plus simplement, que $1 \leq j < n$.

On a vu dans la preuve de terminaison que, dans la boucle externe, on a $i \in [1, n[$. j est initialisé à i puis décroît, donc, à tout instant, $j \leq i < n$.

Pour les accès (ligne 15), ils sont effectués uniquement si $j > 0$ est vérifié, CQFD.

Pour l'appel à echange (ligne 16), il est effectué dans la boucle, donc uniquement si la condition du while est vérifiée, donc en particulier uniquement si $j > 0$ est vérifié, CQFD.

Le résultat est correct

Pour qu'une fonction trie correctement un tableau, il faut :

1. que le tableau final soit une permutation du tableau initial
2. que les éléments du tableau final soient classés par ordre croissant

La première propriété est facile à vérifier : le tableau n'est jamais modifié, sauf éventuellement par la fonction echange qui réalise une permutation du tableau (échange de deux éléments). Une succession de permutations étant une permutation, le tableau final est bien une permutation du tableau initial.

La seconde propriété peut s'écrire plus formellement :

Post-condition 1.

$$\forall k \in [0, n-1[\qquad tab[k] \leq tab[k+1] \tag{1}$$

Elle va être vérifiée par récurrence en utilisant un invariant de boucle.

Considérons l'invariant suivant pour la boucle externe :

Invariant 2. *Au début de la i^e boucle,*

$$\forall k \in [0, i-1[\qquad tab[k] \leq tab[k+1] \tag{2}$$

On peut voir que, si l'invariant 2 est effectivement vérifié, alors la post-condition 1 est valide. En effet, l'état à la fin d'une itération est le même que celui au début de l'itération suivante (sans supposer que le corps de cette itération soit exécuté ou pas). La ligne 12 montre que l'itération extérieure est exécutée $n-1$ fois (i prenant les valeurs $\{1, \cdots, len(tab)-1\}$).

L'état à la fin de la dernière itération (i.e. de l'itération $len(tab)-1 = n-1$) est le même que l'état du début de l'itération suivante (i.e. l'itération n) qui ne sera pas exécutée puisqu'on sortira de la boucle. Mais, avec $i = n$, l'invariant 2 nous donne exactement la post-condition 1.

Montrons la validité de l'invariant 2 pour $i \in [1, n]$ avec un raisonnement par récurrence.

Pour $i = 1$, au début de la première itération, $[0, i-1[$ est l'ensemble vide, donc l'invariant est valide.

Soit $i \in [2, n]$. On suppose que l'invariant est vrai pour $i-1$, montrons qu'il est vrai pour i. Au début de l'itération $i-1$, on a donc

$$\forall k \in [0, i-2[\qquad tab[k] \leq tab[k+1] \tag{3}$$

Comme $i - 1 < n$, le corps de la boucle est exécuté. Pour finir la preuve, il suffit de montrer qu'en exécutant les lignes 13 à 16, on aura

$$\forall k \in [0, i-1[\qquad tab[k] \leq tab[k+1] \tag{4}$$

On va montrer () en introduisant un second invariant pour la boucle interne :

Invariant 3. *Au début de l'itération l de la boucle interne, on a*

$$j = i - (l-1) \tag{5}$$

$$\forall k \in [0, j-2[\qquad tab[k] \leq tab[k+1] \tag{6}$$

$$\forall k \in [j, i-1[\qquad tab[k] \leq tab[k+1] \tag{7}$$

$$Si \quad j > 0 \quad et \quad j < i-1 \quad alors \quad tab[j-1] \leq tab[j+1] \tag{8}$$

Supposons que l'invariant soit vrai à chaque tour de boucle. Lorsque l'on sort de la boucle, c'est parce que la condition de la ligne 15 est fausse :

— si $j = 0$, alors la propriété (7) de l'invariant 3 nous donne exactement ce qu'on veut obtenir, i.e. ;

— sinon, c'est que $tab[j] \geq tab[j-1]$, autrement dit

$$tab[j-1] \leq tab[(j-1)+1] \tag{9}$$

Les propriétés (9), (6) et (7) permettent de conclure que

$$\forall k \in [0, i-1[\qquad tab[k] \leq tab[k+1]$$

En effet, suivant la valeur de k, on utilise l'une des trois propriétés.

Dans les deux cas, on aura montré (), et donc on aura prouvé l'invariant 2 et donc la post-condition 1

Il nous reste donc à montrer que l'invariant 3 est toujours valide à chaque tour de boucle interne.

Au début de la première itération interne ($l = 1$), on a $j = i$ (ligne 14). On en déduit les propriétés :

— (5) immédiat ;

— (6) grâce à (3) (la ligne 14 n'a pas d'effet sur cette propriété) ;

— (7) $[j, i-1[$ est vide ;

— (8) le prédicat de l'implication est faux.

L'invariant 3 est donc valide au début de la première itération interne.

Supposons que l'invariant 3 est valide au début de l'itération l de la boucle interne et que le corps de cette boucle s'exécute. Montrons qu'il est alors encore valide au début de l'itération suivante $l' = l + 1$. Pour plus de clarté, on note avec le symbole $'$ les valeurs des variables à la fin de l'itération l (et on garde la notation sans le symbole $'$ pour les valeurs au début de l'itération l).

On a donc à notre disposition (5), (6), (7) et (8). La condition de la ligne 15 est vraie (puisque la boucle s'exécute), donc on a aussi

$$j > 0 \tag{10}$$

$$tab[j] < tab[j-1] \tag{11}$$

Et, en raison du code des lignes 16 (échange de $tab[j]$ et $tab[j-1]$) et 17 (décrémentation de j), on a :

$$j' = j - 1 \tag{12}$$

$$tab'[j] = tab'[j'+1] = tab[j-1] = tab[j'] \tag{13}$$

$$tab'[j-1] = tab'[j'] = tab[j] = tab[j'+1] \tag{14}$$

$$\forall k \in [0, n[\setminus \{j-1, j\} \qquad tab'[k] = tab[k] \tag{15}$$

Les propriétés (11) et (15) peuvent aussi s'écrire (en utilisant (12), (13) et (14))

$$tab'[j'] < tab'[j'+1] \tag{16}$$

$$\forall k \in [0, n[\setminus \{j', j'+1\} \qquad tab'[k] = tab[k] \tag{17}$$

On début de l'itération suivante, on est dans le même état qu'à la fin de l'itération précédente. Grâce à (12), on a

$$j' = i - (l' - 1) \tag{18}$$

De (6), (12) et (17), on déduit immédiatement

$$\forall k \in [0, j'-2[\qquad tab'[k] \leq tab'[k+1] \tag{19}$$

De (7), (12) et (17), on déduit immédiatement

$$\forall k \in [j'+2, i-1[\qquad tab'[k] \leq tab'[k+1] \tag{20}$$

Si $j'+1 < i-1$, alors $j < i-1$. Sachant en outre (10), en utilisant successivement (13), (8), (15) et (12), on obtient :

$$tab'[j'+1] = tab[j-1] \leq tab[j+1] = tab'[j+1] = tab'[j'+2] \tag{21}$$

Avec (16), (21) et (20), on en déduit que

$$\forall k \in [j', i-1[\qquad tab'[k] \leq tab'[k+1] \tag{22}$$

Enfin, supposons $j' > 0$ et $j' < i-1$. Alors, en utilisant successivement (15), (6) et (13), on a :

$$tab'[j'-1] = tab[j-2] \leq tab[j-1] = tab'[j'+1] \tag{23}$$

Au début de l'itération l', i.e. à la fin de l'itération l, on a bien (18), (19), (22) et (23) qui valident l'invariant 3.
L'invariant 3 étant prouvé, cela termine la preuve de l'invariant 2 et donc de la post-condition 1.
L'algorithme du tri par insertion proposé est donc correct.