Université de Lille 1 SCIENCES ET TECHNOLOGIES

ufr d'IEEA
Formations en
Informatique de
Lille 1

L1 MASS/SESI/PEIP - 2015–2016

Algorithmes et Programmation 1

TD : Tri

Exercice 1 *Ordre sur les dates*

On suppose dans cet exercice que les dates sont représentées par un dictionnaire qui possède trois clés respectivement nommées `'jour'` `'mois'` et `'annee'`. On suppose de plus que les valeurs associées à ces trois clés sont des entiers. On suppose enfin que la validité des données a été testé lors de la création des dictionnaires.

Question 1 Réalisez une fonction `compare_date` paramétrée par deux dates `d1` et `d2`, qui renvoie un entier valant respectivement -1, 0 ou 1 selon que respectivement `d1` est avant `d2`, `d1` égale `d2` , `d1` est après `d2`.

```
>>> compare_date ({'annee': 2016, 'jour': 19, 'mois': 2},{'jour': 1, 'mois': 3, 'annee': 2016})
-1
>>> compare_date ({'annee': 2016, 'jour': 19, 'mois': 2},{'jour': 19, 'mois': 2, 'annee': 2016})
0
>>> compare_date ({'annee': 2017, 'jour': 19, 'mois': 2},{'jour': 19, 'mois': 2, 'annee': 2016})
1
```

Corrigé

```
>>> print(getsource(compare_date))
def compare_date (d1,d2):
    """
    renvoie
    - -1 si la date ``d1`` est avant la date ``d2``
    - 0 si les deux dates ``d1`` et ``d2`` sont égales
    - 1 si la date d1 est après la date ``d2``.

    Exemples :

    >>> compare_date ({'annee': 2016, 'jour': 19, 'mois': 2},{'jour': 1, 'mois': 3, 'annee': 2016})
    -1
    >>> compare_date ({'annee': 2016, 'jour': 19, 'mois': 2},{'jour': 19, 'mois': 2, 'annee': 2016})
    0
    """
    if d1['annee'] < d2['annee']:
        return -1
    elif d1['annee'] > d2['annee']:
        return 1
    elif d1['mois'] < d2['mois']:
        return -1
    elif d1['mois'] > d2['mois']:
        return 1
    elif d1['jour'] < d2['jour']:
        return -1
    elif d1['jour'] > d2['jour']:
        return 1
    else:
        return 0
```


cc BY NC SA

On donne maintenant la variable `etudiants` contenant des quadruplets, `nip` (un entier) , `nom` (une chaine), `prenom` (une chaine), `date_de_naissance` (un dictionnaire comme ci-dessus).

```
>>> etudiants = [ (99998132,"Calbuth","Raymond",{'jour':12,'mois':12,'annee':1987}),
...                (99994451,"Talon","Achille",{'jour':7,'mois':11,'annee':1963}),
...                (99996348,"Calbuth","Monique",{'jour':29,'mois':7,'annee':1987}),
...                (99995433,"Blanc-Sec","Adèle",{'jour':17,'mois':4,'annee':1976}),
...                (99997674,"Brisefer","Benoît",{'jour':15,'mois':12,'annee':1960}),
...                (99998324,"Lagaffe","Gaston",{'jour':28,'mois':2,'annee':1957}) ]
```

Question 2 Donnez une instruction utilisant la méthode `sort` permettant de modifier la variable `etudiants` afin que la liste des étudiants soit rangée par ordre croissant de nip.

Corrigé

```
>>> etudiants.sort ()
>>> for et in etudiants:
...   print (et[1], et[2])
...
Talon Achille
Blanc-Sec Adèle
Calbuth Monique
Brisefer Benoît
Calbuth Raymond
Lagaffe Gaston
```

Question 3 Donnez une instruction permettant de modifier la variable `etudiants` afin que la liste des étudiants soit rangée par ordre décroissant de nip.

Corrigé

```
>>> etudiants.sort (reverse = True)
>>> for et in etudiants:
...    print (et[1], et[2])
...
Lagaffe Gaston
Calbuth Raymond
Brisefer Benoît
Calbuth Monique
Blanc-Sec Adèle
Talon Achille
```

Question 4 Donnez une instruction permettant de modifier la variable `etudiants` afin que la liste des étudiants soit rangée par ordre croissant de nom. (Une fonction étudiée en cours peut s'avérer bien utile)

Corrigé

```
>>> from functools import cmp_to_key
>>> etudiants.sort (key = cmp_to_key (seq_compare_deuxieme))
>>> for et in etudiants:
...    print (et[1], et[2])
...
Blanc-Sec Adèle
Brisefer Benoît
Calbuth Raymond
Calbuth Monique
Lagaffe Gaston
Talon Achille
```

Question 5 Proposez une fonction de comparaison d'étudiants avec laquelle vous pourrez modifier la variable `etudiants` afin que la liste des étudiants soit rangée par ordre croissant de date de naissance.

Corrigé

```
>>> print (getsource(compare_age_etudiants))
def compare_age_etudiants (et1, et2):
    """
    renvoie
    - -1 si l'étudiant et1 est plus jeune que l'étudiant et2
    - 0 si les étudiants et1 et et2 ont même âge
    - 1 si l'étudiant et1 est plus vieux que l'étudiant et2

    CU : aucune

    """
    return compare_date (et1[3], et2[3])

>>> etudiants.sort (key = cmp_to_key (compare_age_etudiants))
>>> for et in etudiants:
...     print (et[1], et[2])
...
Lagaffe Gaston
Brisefer Benoît
Talon Achille
Blanc-Sec Adèle
Calbuth Monique
Calbuth Raymond
```

Exercice 2 *Ordre sur les chaines*

Dans cet exercice, on souhaite définir une variante de l'ordre des caractères puis étendre cette variante aux chaines de caractères. On dit que c est une lettre si c est

- soit entre 'A' et 'Z'
- soit entre 'a' et 'z'

Question 1 Réalisez une fonction `compare_car` paramétrée par deux caractères telle que `compare_car(c1,c2)`

- renvoie soit -1, 0 ou 1
- avec pour les lettres l'ordre défini par `'aAbBcC....yYzZ'`
- tout caractère non lettre étant plus grand que les lettres, et dans leur ordre habituel.

```
>>> compare_car ('a', 'A')
-1
>>> compare_car ('A','b')
-1
>>> compare_car ('@', 'Z')
1
>>> compare_car ('9', '@')
-1
>>> compare_car ('@', '@')
0
```

Corrigé

```
>>> print(getsource(compare_car))
def compare_car (c1, c2):
    """

    CU : aucune

    Exemples :

    >>> compare_car ('a', 'A')
    -1
    >>> compare_car ('A','b')
    -1
    >>> compare_car ('@', 'Z')
    1
    >>> compare_car ('9', '@')
    -1
    >>> compare_car ('@', '@')
    0
    """
    if c1 in LETTRES and c2 in LETTRES:
        return compare_num (LETTRES.index (c1), LETTRES.index (c2))
    elif c1 in LETTRES:
        return -1
    elif c2 in LETTRES:
        return 1
    else:
        return compare_num (ord (c1), ord (c2))
```

Question 2 Réalisez une fonction `compare_chaine` permettant de comparer les chaines en utilisant la fonction `compare_car` précédente, de sorte que
- les chaînes soient rangées par longueur croissante ;
- les chaînes de même longueurs soient rangées par ordre lexicographique induit par `compare_car` .

```
>>> compare_chaine ('','Tim')
-1
>>> compare_chaine ('Timoleon', 'TiM')
1
>>> compare_chaine ('Timoleon', 'TimoLeon')
-1
>>> compare_chaine ('TimoLeon', 'TimoLeon')
0
```

Corrigé

```
>>> print(getsource(compare_chaine))
def compare_chaine (ch1, ch2):
    """
    CU : aucune

    Exemples :

    >>> compare_chaine ('','Tim')
    -1
    >>> compare_chaine ('Timoleon', 'TiM')
    1
    >>> compare_chaine ('Timoleon', 'TimoLeon')
    -1
    >>> compare_chaine ('TimoLeon', 'TimoLeon')
    0
    """
    lg1 = len (ch1)
    lg2 = len (ch2)
    comp_long = compare_num (lg1, lg2)
    if comp_long != 0:
        return comp_long
    else:
        i = 0
        meme_car = True
        while i < lg1 and meme_car:
            meme_car = ch1[i] == ch2[i]
            if meme_car: i += 1
        # (i == lg1 et ch1 == ch2) ou (meme_car != True et ch1[i] != ch2[i])
        if i == lg1:
            return 0
        else:
            return compare_car (ch1[i], ch2[i])
```

Exercice 3 *Un prédicat*

Réalisez un prédicat qui
- prend en paramètre une liste et une fonction de comparaison
- et renvoie `True` si cette liste est triée par l'ordre défini par la fonction de comparaison, et renvoie `False` dans le cas contraire.

```
>>> est_trie ([3,1,4,1,5,9,2], compare_num)
False
>>> est_trie ([1,1,2,3,4,5,9], compare_num)
True
```

Corrigé

```
>>> print(getsource(est_trie))
def est_trie (l, comp):
    """
    renvoie
    - True si la liste ``l`` est triée dans l'ordre défini par
           la fonction de comparaison ``comp``
    - False sinon

    CU: aucune

    Exemple :

    >>> est_trie ([3,1,4,1,5,9,2], compare_num)
    False
    >>> est_trie ([1,1,2,3,4,5,9], compare_num)
    True
    """
    n = len (l)
    i = 0
    while i < n - 1 and comp (l[i],l[i+1]) <= 0:
        i += 1
    return i == n - 1
```

Exercice 4 *Retour sur les doublons*

Proposez un algorithme utilisant les tris pour construire la liste des doublons d'une liste donnée.

Corrigé

1. trier la liste
2. parcourir la liste et ajouter dans la liste de doublons tout élément égal à son suivant

Attention à ne pas introduire de doublons dans la liste des doublons lorsqu'un doublon apparaît plus de deux fois.

Comparez le nombre de comparaisons d'éléments de la liste effectuées par cet algorithme avec celui de l'algorithme que vous avez proposé dans l'exercice de la feuille précédente.

Corrigé

Le nombre de comparaisons effectuées par cet algo est la somme

1. du nombre de comparaisons pour le tri : $\frac{n(n-1)}{2}$ pour le tri par sélection, entre $n-1$ et $\frac{n(n-1)}{2}$ pour le tri par insertion ;
2. et du nombre de comparaisons dans le parcours de la liste triée : $n-1$.

Exercice 5 *Variante du tri par sélection*

Nous avons présenté le tri par sélection du plus petit élément de la tranche restant à trier. Il est possible aussi de faire un tri par sélection du plus grand élément.

Question 1 Donnez l'algorithme de tri par sélection du plus grand élément.

Corrigé

□

Question 2 Implantez cet algorithme pour réaliser une procédure qui trie par cette méthode la liste passée en paramètre.

Corrigé

□

Exercice 6 *Tri à bulle*

L'algorithme 1 est un algorithme de tri dénommé *tri à bulles* qui est une certaine forme de tri par sélection du minimum.

Algorithme 1 Algorithme du tri à bulles

Entrée : t une liste de longueur n.
Sortie : t une liste trié de longueur n contenant les mêmes éléments.

```
1:  pour i variant de 0 à n − 2 faire
2:    {mettre le plus petit élément de la tranche t(i..n − 1) en position i}
3:    pour j variant de n − 1 à i + 1 en décroissant faire
4:      si t(j) < t(j − 1) alors
5:        échanger t(j) et t(j − 1)
6:      fin si
7:    fin pour
8:    {La tranche t(0..i) est triée et ses éléments sont inférieurs ou égaux
      à tous les autres éléments de t.}
9:  fin pour
10: {la tranche t(0..n − 1) est triée.}
```

Question 1 Donnez les états successifs de la liste à la fin de chaque étape de la boucle pour située des lignes 3 à 7 lorsque $i = 0$ et la liste à trier est

$t =$

0	1	2	3	4	5	6	7
T	I	M	O	L	E	O	N

.

Corrigé

□

Question 2 Même question pour la fin de chaque étape de la boucle pour située des lignes 1 à 9.

Corrigé

□

Question 3 Combien de comparaisons d'éléments de la liste sont-elles effectuées lors du tri d'une liste de longueur n ?

Corrigé

□

Question 4 Si lors d'une étape de la boucle pour principale, aucun échange n'est effectué dans la boucle pour interne, c'est que la liste est triée. Il est donc inutile de poursuivre la boucle externe.

Décrivez un algorithme qui tient compte de cette remarque.

Corrigé

```
1: i := 0
2: trie := faux
3: tant que (i < n − 1) et non(trie) faire
4:     trie := vrai
5:     pour j variant de n − 1 à i + 1 en décroissant faire
6:         si t(j) < t(j − 1) alors
7:             échanger t(j) et t(j − 1)
8:             trie := faux
9:         fin si
10:    fin pour
11:    i := i + 1
12: fin tant que
```

Question 5 Combien de comparaisons d'éléments de la liste sont-elles effectuées lors du tri d'une liste de longueur n avec cette variante du tri à bulles ? Décrivez le meilleur et le pire des cas.

Corrigé

□

Question 6 Réalisez une procédure qui trie la liste passée en paramètre selon cet algorithme.

Corrigé

□

Exercice 7 *Tri insertion*

Question 1 Donnez les états intermédiaires successifs de la liste `[4,2,5,3,6,1]` lors de son tri par insertion. Comptez le nombre de comparaisons d'éléments de la liste effectuées pour ce tri.

Question 2 Donnez un encadrement du nombre de comparaisons d'éléments d'une liste de longueur 6 lors du tri par insertion de cette liste.

Exercice 8 *Une variante du tri par insertion*

L'algorithme de tri par insertion vu en cours procède par insertions successives des éléments de la liste dans la tranche de gauche de la liste.

Proposez un algorithme de tri par insertion qui procède par insertions succesives dans la tranche de droite.

Exercice 9

La spécification des fonctions de tri vue en cours les définit comme des procédures qui modifient leur paramètre, tout comme la méthode `sort` en PYTHON.

Reprogrammez les fonctions de tri par sélection et de tri par insertion, de sorte qu'elles ne modifient pas leur paramètre, mais renvoient une nouvelle liste triée.

Exercice 10 *Encore un mélange*

Question 1 Réalisez une procédure nommée `melange` qui modifie la liste passée en paramètre en mélangeant l'ordre de ses éléments. L'algorithme de votre procédure sera une adaptation de l'algorithme du tri par insertion.

```
>>> l = [k for k in range (10)]
>>> melange (l)
>>> l
[4, 2, 1, 9, 0, 8, 3, 5, 7, 6]
```

Corrigé

```
>>> print (getsource(insere_alea))
def insere_alea (l,i):
    """
    insere l'élément d'indice i de l à une position aléatoire choisie entre 0 et i (inclus).

    CU : 1 <= i < len (l)

    """
    aux = l[i]
    pos_insert = randrange (0,i+1)
    k = i
    while k > pos_insert:
        l[k] = l[k-1]
        k -= 1
    l[pos_insert] = aux

>>> print (getsource(melange))
def melange (l):
    """
    modifie de manière aléatoire l'ordre des éléments de la liste l.

    CU : aucune
    Effet de bord : modifie la liste l

    Exemple :

    >>> l = [k for k in range (10)]
    >>> l2 = l.copy ()
    >>> melange (l)
    >>> set ((k, l.count(k)) for k in l) == set ((k, l2.count (k)) for k in l2)
    True
    """
    for i in range (1,len(l)):
        insere_alea (l,i)
```

Question 2 Dessinez l'arbre de tous les déroulements possibles de votre procédure lorsque la liste passée en paramètre est la liste `[1,2,3]`.

Corrigé