Complément au sujet du C.B. d'I.P.T 2016/2017

(Suite et fin du sujet de l'X PSI 2016. Le sujet du C.B. était issu de ce sujet à l'exception des questions de S.Q.L. et de quelques modifications ou indications mineures). La dernière partie ci-dessous fait référence à la notion de `pile` *qui sera introduite dans le cours de deuxième année d'I.P.T. Néanmoins le paragraphe introductif ci-dessous peut suffire pour comprendre ici.*

Partie V. Trier des listes partiellement triées

On souhaite maintenant proposer un algorithme de tri, qui est d'autant plus efficace que la liste donnée en entrée est déjà partiellement triée. On ne donnera pas de définition formelle de ce que ce terme signifie. Dans toute cette partie, pour simplifier, **on ne triera que des listes d'entiers** `(int)`.

Le tri choisi est une version simplifiée du tri utilisé par `Python` (qui s'appelle `TimSort`). On nommera α-tri cette version simplifiée. Ce tri est basé sur un découpage de la liste à trier en séquences croissantes maximales d'éléments consécutifs (appelées *scm*). Ces séquences sont croissantes au sens large. Il consiste à effectuer une succession de fusions de *scm* consécutives jusqu'à n'avoir plus qu'une seule *scm*. Fusionner deux *scm* consécutives consiste à réordonner leurs éléments pour ne former qu'une seule *scm*, comme dans le tri fusion. On notera $|x|$ la longueur d'une *scm* x.

On rappelle qu'une *pile* est une liste pile qui, outre son initialisation, possède deux opérations : l'ajout en fin de liste d'un élément `x` en utilisant `pile.append(x)`, et la suppression du dernier élément en utilisant `pile.pop()`. Cette dernière opération modifie la pile et renvoie l'élément supprimé, ou produit une erreur si `pile` est la liste vide.

L'algorithme α-tri se déroule en deux temps. On commence par partitionner la liste en *scm* consécutives, en identifiant leurs indices de début et de fin dans la liste. Dans un second temps, on effectue les fusions.

Partitionnement en *scm*

Si `s` est une liste d'entiers de longueur $n \geq 1$, son partitionnement en *scm* est l'unique séquence de longueur $k \geq 1$ de couples d'entiers $(d_0, f_0), (d_1, f_1), \dots (d_{k-1}, f_{k-1})$ telle que :

- $d_0 = 0$ et $f_{k-1} = n-1$,
- $d_i \leq f_i$, pour tout $i \in \{0, \dots, k-1\}$,
- $d_{i+1} = f_{i+1}$ pour tout $i \in \{0, \dots, k-2\}$,
- pour tout $i \in 0, \dots, k-1$, la suite $\mathtt{s}[d_i], \mathtt{s}[d_{i+1}], \dots, \mathtt{s}[f_i]$ est croissante au sens large,
- $\mathtt{s}[f_i] > \mathtt{s}[d_{i+1}]$ pour tout $i \in \{0, \dots, k-2\}$.

Exemple : si l'on considère la séquence `s`$= [3, 4, 8, 11, 1, 5, 2, 7, 9, 0, 10, 0]$, on obtient $k = 4$ et la décomposition :

$$\underbrace{3 \leq 4 \leq 8 \leq 11}_{(d_0,f_0)=(0,3)} > \underbrace{1 \leq 5}_{(d_1,f_1)=(4,5)} > \underbrace{2 \leq 7 \leq 9}_{(d_2,f_2)=(6,8)} > \underbrace{0 \leq 10}_{(d_3,f_3)=(9,10)} > \underbrace{0}_{(d_4,f_4)=(11,11)}$$

Question 17.
Écrire une fonction `scm(s)` qui prend une liste `s` en paramètre et renvoie la liste ordonnée des couples d'indices correspondant au partitionnement en *scm* de `s`. Par exemple, avec comme paramètre la liste `s = [3, 4, 8, 11, 1, 5, 2, 7, 9, 0, 10, 0]`, l'appel à `scm(s)` renverra la liste `[(0, 3), (4, 5), (6, 8), (9, 10), (11, 11)]`.

Fusions de deux *scm* consécutives

Les fusions effectuées par α-tri concernent toujours deux *scm* consécutives. Nous aurons donc besoin d'une procédure pour réaliser une telle fusion.

Question 18.
Écrire une procédure `fusionner(s, r1, r2)` qui prend une liste `s` en paramètre ainsi que deux *scm* consécutives encodées par leurs indices de début et de fin, et les fusionne en une seule *scm* : si `r1` $= (d_1, f_1)$ et `r2` $= (d_2, f_2)$, alors après l'appel à la procédure, la partie de `s` située entre les

indices d_1 et f_2 dans `s` doit être triée. Cette procédure ne crée pas une nouvelle liste, elle modifie la liste `s`.

Remarque : Il n'est pas demandé de vérifier que les *scm* sont consécutives. Si nécessaire, on supposera que l'on dispose d'une fonction `copier(s, debut, fin)` qui renvoie une copie de la liste donnée en paramètre entre les indices `debut` et `fin`, que l'on pourra utiliser pour recopier les sous-séquences de `s` qui correspondent aux *scm* décrites par `r1` et `r2`.

Algorithme α-tri

Les fusions des *scm* sont effectuées en deux temps. Tout d'abord, on utilise une pile initialement vide, dans laquelle les *scm* sont ajoutées une par une, dans l'ordre. À chaque fois qu'une *scm* est ajoutée, on compare les longueurs (leurs nombres d'éléments) de la dernière *scm* z de la pile et de l'avant-dernière y (si elle existe). Si $|y| < 2|z|$, on retire y et z de la pile, on les fusionne et on ajoute la *scm* fusionnée dans la pile. On continue à effectuer des fusions tant que la condition sur les longueurs des deux dernières *scm* est vérifiée. Quand on arrive à une pile avec un seul élément, ou telle que $|y| \geq 2|z|$, on ajoute la *scm* suivante dans la pile et on recommence les fusions éventuelles.

Dans un deuxième temps, lorsque toutes les *scm* initiales ont été ajoutées à la pile, on effectue une dernière passe en fusionnant itérativement les deux dernières *scm* de la pile, jusqu'à n'avoir plus qu'une seule *scm*. Cette *scm* est bien la liste initiale triée.

Exemple d'exécution de la phase de fusion pour `[3, 4, 8, 11, 1, 5, 2, 7, 9, 0, 10, 0]` :

```
** Découpage en scm **
Liste à trier: [3, 4, 8, 11, 1, 5, 2, 7, 9, 0, 10, 0]
Liste des scm: [(0, 3), (4, 5), (6, 8), (9, 10), (11, 11)]
** Première phase **
État de la pile: [(0, 3)]
État de la pile: [(0, 3), (4, 5)]
État de la pile: [(0, 3), (4, 5), (6, 8)]
Fusion des scm: (4, 5) et (6, 8). État de la liste: [3, 4, 8, 11, 1, 2, 5, 7, 9, 0, 10, 0]
État de la pile: [(0, 3), (4, 8)]
Fusion des scm: (0, 3) et (4, 8). État de la liste: [1, 2, 3, 4, 5, 7, 8, 9, 11, 0, 10, 0]
État de la pile: [(0, 8)]
État de la pile: [(0, 8), (9, 10)]
État de la pile: [(0, 8), (9, 10), (11, 11)]
** Deuxième phase **
Fusion des scm: (9, 10) et (11, 11). État de la liste: [1, 2, 3, 4, 5, 7, 8, 9, 11, 0, 0, 10]
État de la pile: [(0, 8), (9, 11)]
Fusion des scm: (0, 8) et (9, 11). État de la liste: [0, 0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 11]
État de la pile: [(0, 11)]
La liste triée: [0, 0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 11]
```

Question 19.
À l'aide de la procédure `fusionner` demandée à la question 18 écrire une procédure `depileFusionneRemplace(s, pile)` qui prend en paramètre une liste `s` ainsi qu'une pile de *scm* (sous la forme de couples d'indices de début et de fin). Cette procédure devra retirer les deux *scm* au sommet de la pile, les fusionner dans la liste `s` et replacer les indices de la *scm* fusionnée au sommet de la pile.

Remarque : La pile doit contenir au moins deux *scm* ; on suppose que c'est le cas, et il n'est donc pas demandé de le vérifier.

On rappelle que si `s` est une liste, `s[-1]` et `s[-2]` sont respectivement le dernier et l'avant-dernier élément de la liste, quand ils existent.

Question 20.
En utilisant les questions précédentes, écrire une procédure `alphaTri(s)` qui prend en paramètre une liste `s` et trie cette liste en utilisant l'algorithme α-tri décrit ci-dessus (voir l'exemple). Attention, cette procédure ne crée pas une nouvelle liste, elle modifie la liste passée en paramètre.

L'algorithme `TimSort` *a d'abord été conçu pour le langage* `Python`*. Quelques années après, il a été adopté par d'autres langages de programmation. Il est notamment l'un des tris de la bibliothèque standard du langage* `Java` *depuis la version 7.*

* *
*