TP 3

DES PILES ET DES FILES

La notion de pile repose sur les quatre opérations de base : créer une pile vide, tester si une pile est vide, empiler un élément au sommet de la pile et dépiler l'élément au sommet de la pile (ce qui renvoie sa valeur et l'enlève de la pile). Après avoir exécute le fichier `Pilefile.py` (disponible sur `http://pellerin.xyz`), on pourra utiliser les méthodes de base ainsi :

```
>>> p = Pile()
>>> p.empty()
True
>>> p.empiler(5)
>>> p.empiler(12)
>>> p.empiler(17)
>>> print(p)
Pile de contenu [5, 12, 17]
>>> p.depiler()
17
>>> print(p)
Pile de contenu [5, 12]
>>> p.empty()
False
```

De même, la notion de file repose sur quatre opérations de base : créer une file vide, tester si une pile est vide, ajouter un élément en queue de file (enfiler) et renvoyer l'élément en tête de file en le supprimant (défiler). Le fichier `Pilefile.py` permet d'utiliser les méthodes de base ainsi :

```
>>> f = File()
>>> f.empty()
True
>>> f.enfiler(3)
>>> f.enfiler(6)
>>> f.enfiler(8)
>>> f.enfiler(4)
>>> print(f)
File de contenu [4, 8, 6, 3]
>>> f.defiler()
3
>>> print(f)
File de contenu [4, 8, 6]
>>> f.enfiler(f.defiler())
>>> print(f)
File de contenu [6, 4, 8]
```

Exercice 1

1. Écrire une fonction `consultation(f)` renvoyant la valeur de la tête d'une file `f` sans la modifier.
2. Écrire une fonction `longueurfile(f)` renvoyant la taille d'une file `f` sans la modifier.
3. Écrire une fonction `copiefile(f)` renvoyant une copie d'une file `f` sans la modifier.

Notons que le dernier exercice du TP précédent proposait de programmer des fonctions analogues pour les piles (on s'assurera de savoir le faire !).

Exercice 2

On considère n personnes positionnées en cercle et un entier $k \in [\![2, n-1]\!]$. Ces personnes vont quitter le cercle selon la règle suivante : on choisit une personne en première position et, à partir de cette dernière, on tourne dans le sens des aiguilles d'une montre en éliminant les personnes de k en k, jusqu'à ce qu'il ne reste que $k-1$ personnes dans le cercle.

1. En utilisant la structure de file, écrire une fonction `derniers` d'arguments `f` (file des participants) et `k` et qui renvoie la file finale avec les `k-1` derniers participants.

2. On se place désormais dans le cas $k = 2$ et on numérote les individus de 1 à `n` ; on note J_n le numéro du gagnant.

a. Déterminer la liste des valeurs de J_n pour $n \in [\![2, 20]\!]$.

b. On peut vérifier que, pour tout $n \geqslant 2$, on a $J_{2n} = 2J_n - 1$ et $J_{2n+1} = 2J_n + 1$. En déduire une fonction récursive permettant de calculer J_n.

Exercice 3

On souhaite 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. 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 fondé sur un découpage de la liste à trier en séquences croissantes (au sens large) maximales d'éléments consécutifs (appelées scm) ; 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.

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 $n \geqslant 1$, son partitionnement en scm est l'unique séquence de longueur $k \geqslant 1$ de couples d'entiers $(d_0, f_0), (d_1, f_1), \ldots, (d_{k-1}, f_{k-1})$ telle que :

- $d_0 = 0$ et $f_{k-1} = n-1$,
- $d_i \leqslant f_i$ pour tout $i \in [\![0, k-1]\!]$,
- $d_{i+1} = f_i + 1$ pour tout $i \in [\![0, k-2]\!]$,
- pour tout $i \in [\![0, k-1]\!]$, la suite $\texttt{s}[d_i], \texttt{s}[d_i+1], \ldots, \texttt{s}[f_i]$ est croissante (au sens large),
- $\texttt{s}[f_i] > \texttt{s}[d_{i+1}]$.

Par exemple, pour la séquence `s=[3,4,8,11,1,5,2,7,9,0,10,0]`, on obtient $k = 5$ et :

$$(d_0, f_0) = (0,3),\ (d_1, f_1) = (4,5),\ (d_2, f_2) = (6,8),\ (d_3, f_3) = (9,10) \text{ et } (d_4, f_4) = (11,11).$$

1. Écrire une fonction `scm(s)` qui prend une liste `s` en paramètre et renvoie la liste ordonnée des couples d'indices correspondant au partitionnement de `s` en scm.

2. Écrire une procédure `fusionner(s, r1, r2)` qui prend une liste `s` en paramètre ainsi que deux scm consécutives encodées par les indices de début et de fin, et les fusionne en une seule scm : si `s1` est (d_1, f_1) et `s2` est (d_2, f_2) alors la partie de `s` située entre les indices d_1 et f_2 doit être modifiée et triée après l'appel de la procédure.

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 de la dernière scm z de la pile et de l'avant-dernière y ; si $|y| < 2|z|$ alors on retire y et z, 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 est vérifiée. Quand on arrive à une pile avec un seul élément, ou telle que $|y| \geqslant 2|z|$, 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.

3. À l'aide de la procédure `fusionner`, écrire une procédure `depileFusionneRemplace(s,pile)` qui prend en argument une liste `s` ainsi qu'une pile de scm (sous la formes 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 (on supposera que la pile contient au moins deux scm).

4. Écrire une procédure `alphaTri(s)` qui prend en paramètre une liste `s` et trie cette liste en utilisant l'algorithme exposé ci-dessus.