TP 2

MANIPULATION DE LISTES ET DE PILES

Exercice 1

Dans cet exercice, un segment $[a, b]$ (avec $a \leqslant b$) est représenté par la liste : `[a, b]`.

1. Deux segments sont disjoints lorsque leur intersection est vide. Écrire une fonction `disjoints` de deux arguments `i1` et `i2` qui teste si les segments `i1` et `i2` sont disjoints en renvoyant le booléen `True` s'ils sont disjoints, `False` sinon.

2. La fusion de deux segments a comme minimum le plus petit des minimums des deux segments, et comme maximum le plus grand des maximums des deux segments. Écrire une fonction `fusion` de deux arguments `i1` et `i2` et qui renvoie le segment correspondant à la fusion de `i1` et `i2`.

3. Une « *liste bien formée* » est une liste de segments qui vérifie les propriétés suivantes :

- les segments sont deux à deux disjoints ;
- les segments de la liste sont classés par ordre croissant, en considérant qu'un segment `i1` est strictement plus petit qu'un segment `i2` si et seulement si le maximum de `i1` est strictement inférieur au minimum de `i2`.

a. Les listes suivantes sont-elles bien formées ?

L1 = [[0, 1], [2, 5], [3, 6]], L2 = [[2, 5], [0, 1], [3, 6]], L3 = [[0, 1], [2, 3], [4, 6]].

b. Écrire une fonction `verifie` qui teste si une liste de segments est bien formée (on pourra proposer une version itérative et une version récursive).

4. Écrire une fonction `appartient` de deux arguments `x` et `L` qui teste si la valeur `x` est élément d'un des segments de la liste `L` bien formée.

Exercice 2

Dans cet exercice, on manipule des listes d'entiers et on dira que la liste est «croissante», «décroissante», «monotone» si c'est le cas de la suite d'entiers sous-jacente.

1. Écrire une fonction, de complexité linéaire dans le pire des cas, `estCroissante` d'argument une liste d'entiers et qui renvoie un booléen indiquant si cette liste est croissante.

Écrire de même des fonctions `estDecroissante` et `estMonotone`.

2. Soit la liste `L` = $[u_0, u_1, \cdots, u_{n-1}]$ de longueur n. On appelle tranche de `L` une liste de la forme $[u_i, u_{i+1}, \cdots, u_j]$ où $0 \leqslant i \leqslant j < n$. Écrire une fonction `maxCroissante` d'argument une liste `L` qui renvoie la plus longue tranche croissante de `L`. S'il n'y a pas unicité, on renvoie la première trouvée.

3. Soit la liste `L` = $[u_0, u_1, \cdots, u_{n-1}]$ de longueur n. Une *monotonie* de `L` est un couple d'indices (i, j) où $0 \leqslant i < j < n$ tel que la sous-liste $[u_i, u_{i+1}, \cdots, u_j]$ soit monotone et qu'elle ne le soit plus si on l'étend, à droite ou à gauche, d'un élément supplémentaire (lorsque c'est possible). La monotonie est dite « *banale* » lorsque $j = i + 1$.

a. Proposez une liste d'entiers de longueur 5 qui ne présente que des monotonies banales. Peut-on avoir deux termes consécutifs égaux dans une liste ne présentant que des monotonies banales ?

b. Écrire une fonction `cahots`, de complexité linéaire, qui teste si une liste ne comporte que des monotonies banales.

c. Après avoir créé une liste arbitraire `L` de valeurs distinctes, on peut l'ordonner par `L.sort()`. Imaginer ensuite une méthode pour réordonner `L` de manière à ce qu'elle ne comporte que des monotonies banales.

La notion de pile a été abordée en première année et en cours cette année. On dispose des 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). On peut choisir d'implémenter ces fonctions par exemple à l'aide de listes ou on peut télécharger sur `http://pellerin.xyz` le fichier `Pilefile.py` ce qui permet d'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
```

Exercice 3

Cet exercice propose quelques manipulations de base des piles. Les questions sont indépendantes.

1. Écrire une fonction qui prend une pile en argument et qui renvoie l'élément en bas de la pile.
2. Écrire une fonction qui prend une pile en argument, qui renvoie l'élément en bas de la pile et laisse la pile inchangée.
3. Écrire une fonction qui prend en arguments une pile et un élément et qui renvoie un booléen indiquant si cet élément est contenu dans la pile.
4. Écrire une fonction qui prend une pile en argument et qui renverse la pile (les éléments sont placés à l'envers dans le même ordre).
5. Écrire une fonction renvoyant une copie d'une pile.