Soutien en algorithmique et programmation

Séance 11 : manipulations récursives de listes chainées

Introduction

On va écrire dans cette séance des fonctions **récursives** pour manipuler des listes chainées. On travaillera sur des listes chainées simples sans élément fictif en tête, en utilisant la classe `Cellule` habituelle :

```
class Cellule:
    """
      Une cellule est composee d'une valeur et d'un pointeur vers la
        cellule suivante (ou None s'il n'y a pas de suivant)
    """
    # pylint: disable=too-few-public-methods

    def __init__(self, val, suiv):
        """
          Constructeur
        """
        self.val = val
        self.suiv = suiv

    def __str__(self):
        """
          Afficheur
        """
        return f"{self.val} -> "
```

Pour chaque question, on réfléchira en identifiant à chaque fois le ou les cas de base (c'est à dire le ou les cas pour lequels la récursivité s'arrêtera) et l'hypothèse de récurrence (c'est à dire la façon dont on traitera le cas N en fonction du cas N-1), comme illustré pour la fonction de base fournie ci-dessous.

Création de la liste à partir d'un tableau d'entiers

- cas de base : si le tableau est de taille nulle alors la liste est vide ;
- hypothèse de récurrence : si L' est la liste créée à partir de tous les elements du tableau sauf le premier, alors L est la liste créée en insérant le premier element du tableau en tete de L'.

```
def creer(tab):
    """
      Construit la liste en inserant les valeurs du tableau.
      Les elements doivent apparaitre dans le meme ordre.
      Renvoie la liste construite.
      Cas de base :
        - si le tableau est de taille 0 alors la liste renvoyee est vide
      Hypothese de recurrence :
        - si L' est la liste créée à partir de tous les elements du tableau
          sauf le premier alors L est la liste créée en insérant
          le premier element du tableau en tete de L'
    """
    if not tab: # equivalent a len(tab) == 0
        return None
```

```
    liste_p = creer(tab[1:])
    return Cellule(tab[0], liste_p)

def main():
    """
      Fonction principale
    """
    for taille in range(8):
        print(f"-- Taille = {taille} --")
        tab = [randint(0, 9) for _ in range(taille)]
        print("Tableau initial :", tab)
        liste = creer(tab)
        print("Liste initiale  : ", end="")
        afficher(liste)
        liste1, liste2 = separer(liste)
        print("Listes separees :")
        print("  ", end="")
        afficher(liste1)
        print("  ", end="")
        afficher(liste2)
        liste = trier(creer(tab))
        print("Liste triee     : ", end="")
        afficher(liste)
        print()
    # tests de la fonction fusionner avec des listes deja triees
    tabs = (([], []), ([], [1]), ([2], []), ([1], [2]), ([1, 3], [2]),
            ([3], [2, 4]), ([1, 5, 7], [2, 4, 6, 8]))
    for tab1, tab2 in tabs:
        print(f"Fusion de {tab1} et {tab2} :")
        liste = fusionner(creer(tab1), creer(tab2))
        print("  liste fusionnee : ", end="")
        afficher(liste)
        print()
```

Affichage d'une liste chainée

Ecrire une fonction `affichage(liste)` affichant le contenu de la liste chainée.

Découpage de la liste en deux sous-listes

Ecrire une fonction `separer(liste)` qui découpe la liste en deux sous-listes composées des mêmes éléments que la liste et dans le même ordre, en répartissant un élément sur deux dans chaque sous-liste. La fonction renvoie les deux sous-listes à la fin sous la forme d'un tuple.

Fusion de deux listes triées

Ecrire une fonction `fusionner(liste1, liste2)` qui prend deux listes triées par ordre croissant en paramètre, et renvoie la liste triée par ordre croissant constituée des éléments des deux sous-listes.

Tri d'une liste chainée

Ecrire une fonction `trier(liste)` qui trie la liste chainée passée en paramètre en implantant l'algorithme suivant :

- on sépare la liste en deux sous-liste liste1 et liste2 ;
- on trie les sous-listes liste1 et liste2 ;
- on fusionne les sous-listes liste1 et liste2 pour produire la liste triée.