Soutien en algorithmique et programmation

Séance 3 : manipulations de tableaux

Introduction

Pendant cette séance, on va manipuler une structure de données très classique : un tableau. Python est un des rares langages à ne pas fournir cette structure de données, mais en propose une autre qui lui ressemble beaucoup : les vecteurs (appelés *list* dans la spécification en anglais du langage, mais on évitera ce terme pour ne pas confondre avec les listes chainées que l'on verra bientôt en cours).

La différence entre un tableau et un vecteur est qu'un tableau a une taille fixe alors qu'un vecteur peut grandir ou rétrécir pendant l'exécution du programme. Pour simuler des tableaux en Python, on s'interdira donc d'utiliser les fonctions permettant de modifier la taille d'un vecteur (notamment `append` et `del`).

Manipulations de base

Vous implanterez les fonctions ci-dessous dans un fichier `tableaux.py` dont on donne la fonction principale (pour tester au fur et à mesure les fonctions implantées) :

```
def main():
    """
      Fonction principale
    """
    for taille in range(6):
        print("-- Taille =", taille, "--")
        tab = creer(taille)
        print("Tableau initial :", tab)
        for idx in range(taille):
            print("Indice de", tab[idx], ":", indice_prem_occ(tab, tab[idx]))
        print("Indice de 10 :", indice_prem_occ(tab, 10))
        inverser(tab)
        print("Tableau inverse :", tab)
        print()
```

Initialisation du tableau

Implantez une fonction `creer(taille)` qui renvoie un tableau de `taille` entiers tirés aléatoirement entre 1 et 9. On rappelle qu'il existe en Python une facilité syntaxique permettant de remplir un tableau sans écrire explicitement une boucle. Par exemple, le code `[x*x for x in range(10)]` crée et remplit un tableau avec les carrés de 0, 1, 2, ... 9.

Pour générer des nombres aléatoires, on utilisera une fonction `randint(inf, sup)` qui renvoie un entier aléatoire dans l'intervalle [inf .. sup]. On pourra l'inclure en ajoutant la ligne `from random import randint` en haut de notre programme.

Recherche d'un élément dans le tableau

Implantez une fonction `indice_prem_occ(tab, val)` qui renvoie l'indice de la première occurence de `val` dans le tableau `tab`, ou -1 si le tableau ne contient pas `val`.

Echange de deux éléments

Implantez une fonction `echanger(tab, idx1, idx2)` qui échange les éléments d'indices `idx1` et `idx2` dans le tableau.

Inversion du tableau

Implantez une fonction `inverser(tab)` qui inverse les éléments du tableau en utilisant la fonction `echanger` précédente. Par exemple, `inverser([1,2,3,4,5])` renverra le tableau `[5,4,3,2,1]`.

Implantation d'une pile bornée

On va maintenant utiliser un tableau pour implanter une structure de données fréquemment utilisée : une pile bornée. Une pile est une structure de données implantant une politique « premier entré, dernier sorti » : si on empile des assiettes les unes au dessus des autres, on doit d'abord enlever les assiettes du dessus (celles empilées en dernier) avant d'accéder à celles du dessous (celles empilées en premier).

On va construire une pile bornée, c'est à dire avec une taille maximale fixe. On définira donc une constante `TAILLE_PILE = 5` tout en haut de notre programme.

On pourra tester les fonctions qu'on va écrire avec le programme principal ci-dessous. On donne aussi deux fonctions de base pour comprendre comment implanter la pile.

Création de la pile

On a d'abord écrit une fonction `creer()` qui renvoie une nouvelle pile. On voit qu'on a choisi de représenter la pile sous la forme d'un tableau dans lequel :

- les cases d'indices 1 .. TAILLE_PILE contiendront les valeurs stockées dans la pile ;
- la case d'indice 0 contient l'indice du **sommet de pile** : le sommet de pile est l'indice de la case contenant la dernière valeur empilée, ou 0 par convention quand la pile est vide comme c'est le cas lorsqu'on la crée.

On note aussi qu'on remplit le tableau par défaut avec le caractère '?' : ce caractère n'a aucun sens ici, mais on est obligé d'initialiser le tableau avec valeurs, donc on met n'importe-laquelle.

Affichage de la pile

On donne aussi la fonction `afficher(pile)` pour afficher le contenu de la pile. On voit qu'on défini une constante `IX_SOM` qui contient l'indice de la case contenant le sommet de pile, pour rendre le code plus lisible. Il faut donc définir cette constante tout en haut de notre programme : `IX_SOM = 0`.

```
# tab[0] = sommet : l'indice de la derniere valeur empilee, 0 quand la pile est vide
# tab[1:] = les valeurs contenues dans la pile
def creer():
    """
      Crée une pile de taille donnée
    """
    # On doit bien initialiser les cases du tableau avec des valeurs, on choisit donc de
    #   mettre un '?' mais on ne doit jamais lire le contenu d'une case avant d'y avoir
    #   affecté une valeur significative
```

```
    return [0] + ['?' for _ in range(TAILLE_PILE)]

def afficher(pile):
    """
      Affiche le contenu de la pile
    """
    print(f"{pile} (sommet = {pile[IX_SOM]}, valeurs = {pile[IX_SOM + 1:]}")

def main():
    """
      Programme principal
    """
    pile = creer()
    afficher(pile)
    for _ in range(TAILLE_PILE + 1):
        val = randint(1, 9)
        print("Valeur à empiler :", val)
        empiler(pile, val)
        afficher(pile)
    for _ in range(TAILLE_PILE + 1):
        val = depiler(pile)
        if val:
            print(f"Valeur dépilée = {val}")
            afficher(pile)
```

Empiler une valeur dans la pile

On doit maintenant implanter une fonction `empiler(pile, val)` qui empile la valeur `val` dans la pile passée en paramètre. Cette fonction doit donc :

1. vérifier que la pile n'est pas déjà pleine, et si c'est le cas afficher un message d'erreur et sortir de la fonction sans rien faire d'autre ;
2. incrémenter le sommet de pile, puisqu'on va ajouter une nouvelle valeur ;
3. écrire la valeur dans le tableau à l'indice du nouveau sommet de pile.

Dépiler une valeur de la pile

On va maintenant implanter l'opération inverse, c'est à dire dépiler la valeur la plus récente de la pile. Cette fonction doit donc :

1. vérifier que la pile n'est pas vide, et si c'est le cas afficher un message d'erreur et renvoyer la valeur `None` qui est une valeur générique qu'on renvoie en Python quand on n'a rien d'autre à renvoyer ;
2. récupérer la valeur la plus récente de la pile dans une variable locale ;
3. décrémenter le sommet de pile puisqu'on vient d'enlever une valeur de la pile ;
4. renvoyer la valeur qu'on avait stockée dans une variable.

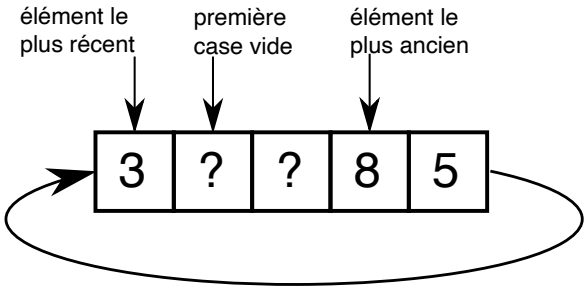
On voit ici qu'on ne va pas écraser la valeur retirée du tableau : la mémoire contiendra toujours cette valeur en pratique. C'est donc le sommet de pile qui permet de savoir quel est l'intervalle de valeurs significatives dans le tableau.

Implantation d'une file FIFO bornée

On va finalement implanter une autre structure de données classique : une file FIFO (*First-In, First-Out*) qui fonctionne selon le principe un peu inverse d'une pile, c'est à dire selon la politique « premier entré, premier sorti » : on insère les éléments d'un côté et on les retire de l'autre.

Comme pour la pile, on va utiliser un tableau pour stocker les valeurs. Mais ce tableau sera ici géré selon

le modèle du « tableau circulaire » : cela signifie que si on arrive au bout du tableau mais qu'il y a des cases libres au début (car on a retiré des valeurs anciennes), alors on utilisera les cases du début lorsqu'on aura besoin d'insérer de nouvelles valeurs. Le dessin ci-dessous illustre ce principe de tableau circulaire :



Comme pour la pile, la file aura une taille fixe : on définira donc une constante `TAILLE_FILE = 5` en haut du programme.

On pourra tester les fonctions qu'on va écrire avec le programme principal ci-dessous. On donne aussi deux fonctions de base pour comprendre comment implanter la file.

Création de la file

On commence par écrire la fonction `creer` qui renvoie une nouvelle file. On voit qu'on a choisi de représenter la file sous la forme de deux tableaux :

1. le premier contient tout simplement les valeurs stockées dans la file, dans les indices [0 .. TAILLE_FILE-1] ;
2. le deuxième va contenir des «méta-informations» nécessaires à la gestion de la file, et plus précisément :
    1. l'indice du premier élément de la file (c'est à dire le plus ancien), qui sera stocké à l'indice 0 par convention ;
    2. le nombre d'éléments dans la file, qui sera stocké à l'indice 1 par convention.

On aura donc besoin de définir deux constantes au début du programme pour accéder à ces méta-informations de façon propre :

```
IX_PREM = 0
IX_NBR = 1
```

Affichage de la file

On donne aussi la fonction d'affichage de la file qui va utiliser les constantes définies ci-dessus.

```
def creer():
    """
      Crée une file de taille donnée
    """
    return ['?' for _ in range(TAILLE_FILE)], [0, 0]

def afficher(tab, info):
    """
      Affiche le contenu de la file
    """
    print(f"(premier = {info[IX_PREM]}, nombre = {info[IX_NBR]}) {tab}")

def test_inserer(tab, info, vals):
```

```
    """
      Test d'insertions
    """
    for val in vals:
        print("Valeur à insérer :", val)
        inserer(tab, info, val)
        afficher(tab, info)

def test_retirer(tab, info, nbr):
    """
      Test de suppressions
    """
    for _ in range(nbr):
        val = retirer(tab, info)
        if val:
            print(f"Valeur retirée = {val}")
            afficher(tab, info)

def main():
    """
      Programme principal
    """
    tab, info = creer()
    afficher(tab, info)
    test_retirer(tab, info, 1)
    test_inserer(tab, info, (1, 2, 3))
    test_retirer(tab, info, 2)
    test_inserer(tab, info, (4, 5, 6))
    test_retirer(tab, info, 1)
    test_inserer(tab, info, (7, 8, 9))
```

Insertion d'un élément dans la file

On doit maintenant implanter la fonction `inserer(tab, info, val)` qui va insérer la valeur `val` dans la file, représentée par ces deux tableaux (`tab` qui contient les valeurs et `info` les méta-informations).

Là-encore, cette fonction doit d'abord vérifier que la file n'est pas pleine, avant de faire l'insertion à proprement parler et de mettre à jour les méta-informations. On devra bien implanter la politique du tableau circulaire comme expliqué plus haut.

Suppression d'un élément dans la file

On va enfin implanter la fonction `retirer(tab, info)` qui doit renvoyer la valeur la plus ancienne présente dans la file.

On doit donc bien vérifier que la file n'est pas vide, puis renvoyer la valeur et mettre à jour bien sûr les méta-informations.